Second Interim Report

NASA CR 86256

RESEARCH IN THE EFFECTIVE IMPLEMENTATION OF GUIDANCE COMPUTERS WITH LARGE SCALE ARRAYS

BY J. J. Pariser
F. D. Erwin
J. F. McKevitt
J. A. Burke
C. P. Disparte

Submitted to

Electronics Research Center

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION

July 1969

(Revised September 1969)

FR 69-11-1000

Prepared under Contract No. NAS 12-665 by
Hughes Aircraft Company
Fullerton, California

Dr. Harold E. Maurer
Technical Monitor
Electronics Research Center
Cambridge, Massachusetts

Requests for copies of this report should be referred to:

NASA Scientific and Technical Information Facility
P. O. Box 33, College Park, Maryland 20740

Second Interim Report

NASA _____

RESEARCH IN THE EFFECTIVE IMPLEMENTATION OF GUIDANCE COMPUTERS WITH LARGE SCALE ARRAYS

BY J. J. Pariser
F. D. Erwin
J. F. McKevitt
J. A. Burke
C. P. Disparte

July 1969
(Revised September 1969)

FR 69-11-1000

Prepared under Contract No. NAS 12-665 by
Hughes Aircraft Company
Fullerton, California

Electronics Research Center

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION

FOREWORD

This Second Interim Report summarizes the overall study results and updates the First Interim Report submitted October, 1968 and revised May, 1969. The study has resulted in building blocks for effective implementation of guidance computers with large scale arrays. The study is being conducted by Hughes-Fullerton under Contract No. NAS 12-665 for the Electronics Research Center of the National Aeronautics and Space Administration.

CONTENTS

ILLUSTRATIONS

ILLUSTRATIONS (Continued)

ILLUSTRATIONS (Continued)

BLANK PAGE FOLLOWS vii/viii

SECTION 1
SUMMARY AND INTRODUCTION

SUMMARY OF RESEARCH PERFORMED TO IMPLEMENT GUIDANCE COMPUTERS WITH LARGE SCALE ARRAYS

The NASA Modular Computer Breadboard (MCB) has been designed by selection and microprogramming of functional logic characters which have been partitioned to minimize large scale integration (LSI) chip types. Alternative implementations, both of the characters and of the MCB, have been made with favorable results.

This study applies functional logic characters to the selected system design of the NASA Modular Computer (NMC) as represented by a modular computer breadboard (MCB). The characters are specified as the building blocks and a repertoire of microprogram instructions is defined. The character set totals ten unique building blocks, which are interconnected as shown in the figure to form digital equipments. A micromemory word of 50 bits is specified. It may be utilized either singly or in pairs, each containing two instructions and a constant field. Based on available information of the MCB Computer, a design of each of the five units was completed using the functional characters. The design of each unit in general includes selecting proper characters and microprogramming the operation of each character to provide all specified MCB machine instructions and operations. Implementation of the MCB using functional characters is evaluated resulting in favorable operating time, reduced pins, reduced total cards, and reduced card types.

Since the characters represent a logical complement, they can be implemented with almost any type of circuitry that meets the speed-power considerations. The study is proceeding utilizing T^2L circuits as stipulated by NASA. However, Hughes strongly recommends that MOS circuits receive parallel consideration as the implementation vehicle. The decision does not need to be made until the commencement of a prototype design of a computer. Ion implantation or a similar speed-up is required if MOS is to be used to meet the speed and power objectives. Hughes considers the reliability and availability risks reasonable and manageable.

Hughes also strongly recommends that the restriction of about 100 gates per chip be removed and that the gate per chip size be made as large as possible commensurate with anticipated production capabilities of the 1970's. In our opinion, the larger gate per chip implementation will result in improved reliability.

A significant portion of this report is devoted to demonstrating the application of a minimal set of general-purpose functional characters (logic arrays) to the implementation of an existing logic design of a general-purpose computer (the MCB). The reader is therefore introduced first to the set of logical building blocks. He is then presented with a repertoire of micro-instructions and microprogram-word structure suitable for controlling the desired logic functions of the specified machine instructions. Following the presentation of this basic background material, the detailed preliminary implementation of the MCB Computer is given. In general, each unit of the MCB is implemented with the functional characters as shown in a block diagram form. A flow diagram presents the microprogram operations for general unit operation and, in some cases, for specific detailed examples. Some general observations and conclusions related to the functional character implementation are presented. View graphs shown at NASA ERC as part of the first presentations are reproduced in the appendix accompanied by a few appropriate comments.

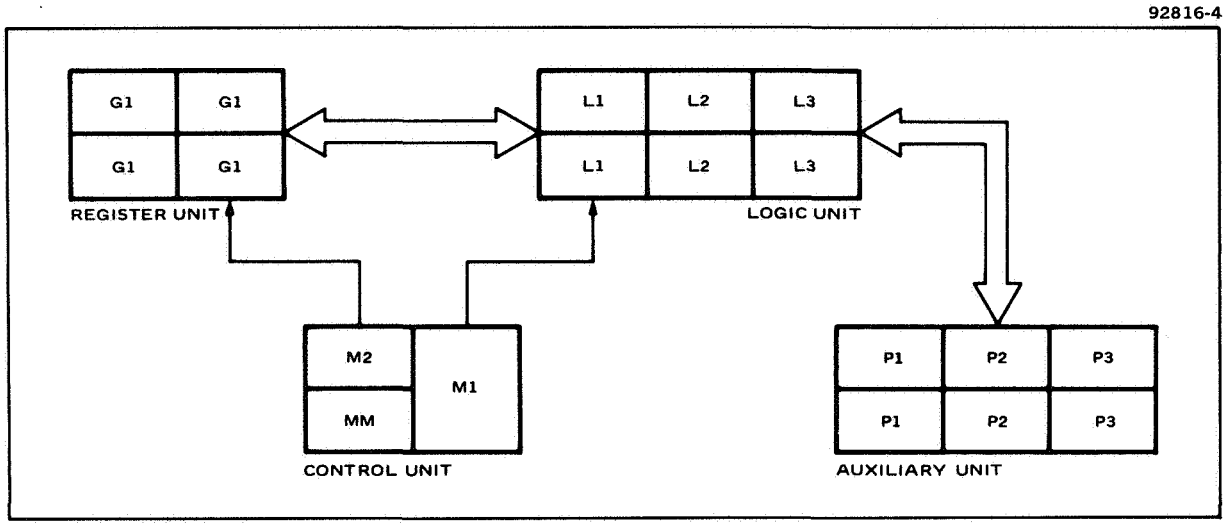


Figure 1-1. Typical Functional Character Configuration. The 10 character types defined for the MCB are grouped into common units, as shown here, in this computer design method.

ACCOMPLISHMENTS DURING PERIODS OF FIRST AND SECOND INTERIM REPORTS

Revision of the MCB design; refinement of the functional character set, and microprogram repertoire; comparison of cellular arrays with functional characters; and subpartitioning of the characters are the major extensions to the first interim report.

The First Interim Report (submitted October, 1968 and revised May, 1969), presented initial results of the study program. The areas discussed in that report are summarized in the facing table. This Second Interim Report does not attempt to include all of the material of the first except as it relates to work accomplished since its publication. Thus no discussion is given here of implementation of special purpose processors, survey of spaceborne computer characteristics, weighted design considerations for aerospace computers, and review of LSI technology for aerospace computers.

The relationship between First to the Second reports is clarified by the following summary statement which was made in the first report.

"The reader is advised to view this report and effort with its purpose in mind. Namely to demonstrate the feasibility (not practicability) of implementing various digital systems using predesigned functionally-organized logical groupings - functional character set. The fact that the MCB implementation resulted in comparable speeds with the custom-designed logic is indeed an unexpected benefit, since no optimization of the architecture for use of functional characters has been attempted. The purpose of the implementation was simply to demonstrate conclusively that purely on a logical basis equipments can be designed with pre-interconnected logic. Our intent was to completely ignore the circuit aspects (speed, fan-in, fan-out, etc.). Now that the feasibility of designing digital equipments using macro-logical elements (functional characters) has been established, we will proceed according to our work plan with the refinements of the character set and procedures in order to obtain the most efficient design in terms of production risk, reliability, performance, etc."

Since then, more detailed considerations of the design characteristics (both logical and circuits) of the characters have been made. In this report, Section 2 contains basic material first presented in the First Interim Report. Considerable study effort has been spent subsequent to that report in refinements and verification in those areas. However, changes in the character set and microprogram repertoire were few. L3 is the only character having major change; the L3 capabilities were greatly extended. Minor changes were made to the repertoire, and several changes or refinements were made in the character implementation of the NASA MCB computer. A general reprogramming of all microprograms was necessary to increase execution speed and to more efficiently use ROM.

To incorporate the new L3 character, unit interfaces were redesigned, resulting in faster inter-unit communication. The work sharing between the Control Unit and Arithmetic Unit was changed to decrease execution time for some instructions. In addition, the effects of combining the functions of the Control and Arithmetic Units into a single unit were studied.

The MCB implementation was improved, both in performance and character utilization, over the preliminary results.

Alternative schemes for sub-partitioning all characters have been analyzed. The subpartitions contain about 100 gates per chip as opposed to 200 to 400 gates per character with a typical 50-percent degradation in the gate-to-pin ratio. The relative efficiency

of cutpoint and cobweb cellular logic implementation of the character set as opposed to more conventional LSI techniques was studied and no significant advantages in using cellular arrays rather than master slice arrays were found.

Areas Reported in First Interim Report

- Description of Building Blocks
- Description of Microprogram Repertoire
- Preliminary Implementation of the MCB Computer
- Implementation of Special Purpose Computers with the Functional Characters
- Survey of Spaceborne Computer Characteristics
- Weighted Design Considerations for Aerospace Computers
- Review of LSI Technology for Aerospace Computers

Areas Reported in Second Interim Report

- Description of Refined Building Blocks
- Description of Refined Microprogram Repertoire
- Revised Implementation of the MCB Computer
- Explored Combining CU and AU Units of the MCB
- Evaluation and Significance of the Character Set in the MCB Implementation
- Subpartitioning of the Character Set
- Evaluation of Cellular Arrays as Candidates for LSI
- Consideration of Circuit Characteristics

NEED FOR UTILIZATION OF LARGE SCALE INTEGRATION (LSI)

A primary goal of this program is to prove that a judicious partitioning of digital systems, divorced from bias toward any particular system, can result in a set of LSI devices that can entirely implement many different schemes for use of LSI computers and other digital devices.

Since the advent of LSI technology, several schemes have evolved for the utilization of large arrays to their full potential. A common and straightforward approach involves the designer restricting himself to the equipment being designed at the moment. Faced with only a limited set of problems, it is not difficult to specify a small number of LSI array types which will efficiently complete the design. While the results are quite encouraging for specific cases, the drawbacks of any mass adoption of these techniques are obvious. This, the so-called "custom approach," would require the semiconductor manufacturer to be responsive to each customer with numerous low-output production runs of highly specialized devices. The per-unit cost to the user, for his own efforts as well as those of the manufacturer, would be quite high due to the inability to spread initial costs over many devices. In addition, the complexity of 100-gate-plus arrays is such that it is difficult to substitute one for another (with efficient results). This would severely limit the off-the-shelf capabilities of both user and manufacturer.

An obvious solution to these problems is the introduction of a small set of standard LSI chips. Semiconductor suppliers, making tentative advances into LSI product marketing, have already proposed such devices as adders, counters, and shift registers. However, this does not represent the solution to the general problem. A design heavily committed to the use of these devices must fall back on MSI or standard IC for the large remainder of the circuitry. The reason is that adders, counters, registers and other orderly, well-defined areas represent the regions of the system with the highest gate-to-pin ratios. After these portions are lifted out of the system, the remainder is characterized by very low gate-to-pin ratios (notably control and data routing functions). Unable to satisfy the LSI design criteria of high gate-to-pin ratios any longer, the designer must look to more standard components. Unfortunately, any proposed solution to the LSI partitioning problem which lacks a total system approach tends to drift towards this pitfall.

Bit Slicing Versus Functional Partitioning — Researchers striving towards partitioning for total or near-total LSI implementation tend to diverge along one of two conceptual paths; bit-slicing and functional partitioning. To illustrate the difference, consider the data portion of the computer. In functional partitioning one may specify an adder as one LSI array, registers as another, a shift register as a third, and so forth. On the other hand, in bit-slicing one would design an LSI array consisting of a combined one- or two-bit adder, registers, shift registers, etc., then build up his system from this chip type according to the desired word length.

The bit-slice approach has resulted in some notable advantages, particularly the ability to achieve very high gate-to-pin ratios and implement systems using a small number of different array types. However, bit-sliced modules have the basic flaw of being system-dependent. This means that behind such bit-slicing approaches there lie systems, real or implied, for which the resulting arrays are most efficient. An attempt to apply the arrays to a significantly different system results in a poor design. Considering the types of bit-slice devices being proposed, inefficiencies would most often be manifest in the design of a simple device in which the majority of the gates of the array intended to accomplish complex functions are wasted. Although this may be acceptable in some

situations, it is unlikely that it would satisfy the strict requirements of size, weight, power, and reliability imposed by aerospace and military systems.

Standardization of LSI Through Functional Characters — The resulting group of arrays, referred to as a "character set" and each one individually as a different "character", is sufficiently small in number (10), with each type having acceptable size and gate/pin ratio, to be considered acceptable and desirable in view of its wide range of applications. These building blocks are referred to as characters because of the metaphor that may be made between the building blocks and characters of the alphabet (letters). Letters form words to express the language whereas building blocks form units to build the machine. In both cases a closed set (of characters) is used to produce the desired end.

Although the character set is neither rigidly functionally-partitioned nor bit-sliced, it is biased towards functional partitioning to give it the versatility to efficiently implement both complex and simple digital devices. As an approach, functional partitioning has a detailed and successful background. Bit-slicing considerations give the character set its ability to implement systems of varying word lengths.

In addition to providing the user with a standard set of chips to implement many different digital machines, the completeness of the approach (the ability of the characters to implement the whole machine) relieves the user of the burden of logic design. These tasks are reduced to the selection of character types and word lengths.

SECTION 2
DESIGN AND USE OF THE FUNCTIONAL CHARACTER SET

Section 2 — Design and Use of the Functional Character Set
Subsection 1 — Computer Building Blocks

COMPUTER DESIGN BY CONFIGURING BUILDING BLOCKS

The motivation for the functional character set is the recognition that digital equipments of all types consist of relatively few functions. The translation of these functions into specific partitions is the major contribution of the character set development.

Logical Design of digital equipments is based primarily on Boolean logic with some sprinkling of switching theory. However, by and large, the design is an individual matter, lacking standardization, with each designer retracing the steps of his colleague. The methods utilized in programming such as use of macros, higher order languages, etc., are relatively unknown in logical design. This study endeavors to introduce standardization to the task of logic design in a form characteristic of a higher order language.

Standardization is possible because digital devices are constructed of a few basic types of functions — memory, registers, gating, and control. To achieve this standardization, a set of functional characters has been designed and successfully tested for sufficiency of implementing various digital equipments. "Implementation" referred herein and throughout the report implies in the logical sense and not in the circuit sense. A character in this report refers to a logical entity which can be subpartitioned to suit the physical implementation requirements. The work associated with transforming the logical specifications to a physical realization involves many tradeoffs and interactions of various disciplines, and this effort is beyond the scope of the study.

The functional character set is a group of logic arrays forming a self-sufficient family of blocks which reduce computer design to a determination of character types and number. For the design of the MCB, 10 character* types are required as listed below. They are described in greater detail in the following topics.

G1	Micromemory register storage
L1	General logic
L2	Arithmetic logic
L3	Input/Output
M1	Micromemory counter
M2	Micro-instruction Register
MM	Micro-array
P1	Scratch pad memory
P2	Up/Down counter
P3	Switch

The table summarizes the number of gates and pins required for each character. The high gate to pin ratios achieved are an important indicator of success in LSI implementation.

*The words "character" and "card" are used interchangeably. (It is not to be inferred that the logic content of a functional character will necessarily represent the contents of a circuit card.)

TABLE 2-I. COMPOSITION OF THE TEN CHARACTER TYPES SUFFICIENT FOR BUILDING SPECIAL-PURPOSE AND GENERAL-PURPOSE DIGITAL EQUIPMENTS

Character	Function	Gates	Pins	Gate to Pin Ratio
G1	General Register	224	62	3.4
P1	Scratch Pad	Depends on system architecture		
		(8 x 16) bits/block		
L1	Boolean	274	145	1.8
L2	Arithmetic	250	77	3.3
L3	I/O	377	149	2.5
M1	Sequencer	348	91	3.8
M2	Instruction	323	131	2.5
MM	Array	Depends on size of program		
		2048 bits/block		
P2	Up/Down Counter	147	81	1.8
P3	Switch	210	118	1.8

DESCRIPTION OF THE G1 CHARACTER

Each G1 character provides efficient centralized storage for general utilization in varying applications.

The G1 character provides the bulk of storage for operands of the microprogram. Each character contains 4 registers of 8 bits each accompanied by reading and writing selector gates. The storage element is provided with simultaneous dual reading and writing capability. The storage flip flop itself is designed for minimum read after write delay.

Each of the two input busses is common to all registers and carries to the G1 character 8 lines per bus, one line from each bus for each bit of the register (see facing figure). Input data selection is accomplished at the memory element by a coincidence of positive information on a particular input bus and register selection for that bus by destination decoding logic within the character. The destination decoding logic is duplicated to provide for writing from the two input busses into the same character under control of two different microcommands. As will be illustrated later, this is a key factor for the machine expandability property of the character set as it allows G1 to form a data path link between individual logic units under control of up to 2 different micromemories. Different registers in the character may be written into simultaneously.

Reading of the register is provided by dual source decoding logic which gates data to independent dual output busses. This duality provides for information from any two registers to be simultaneously placed on two output busses. Several G1 characters placed in parallel provide registers of more than 8 bits in length.

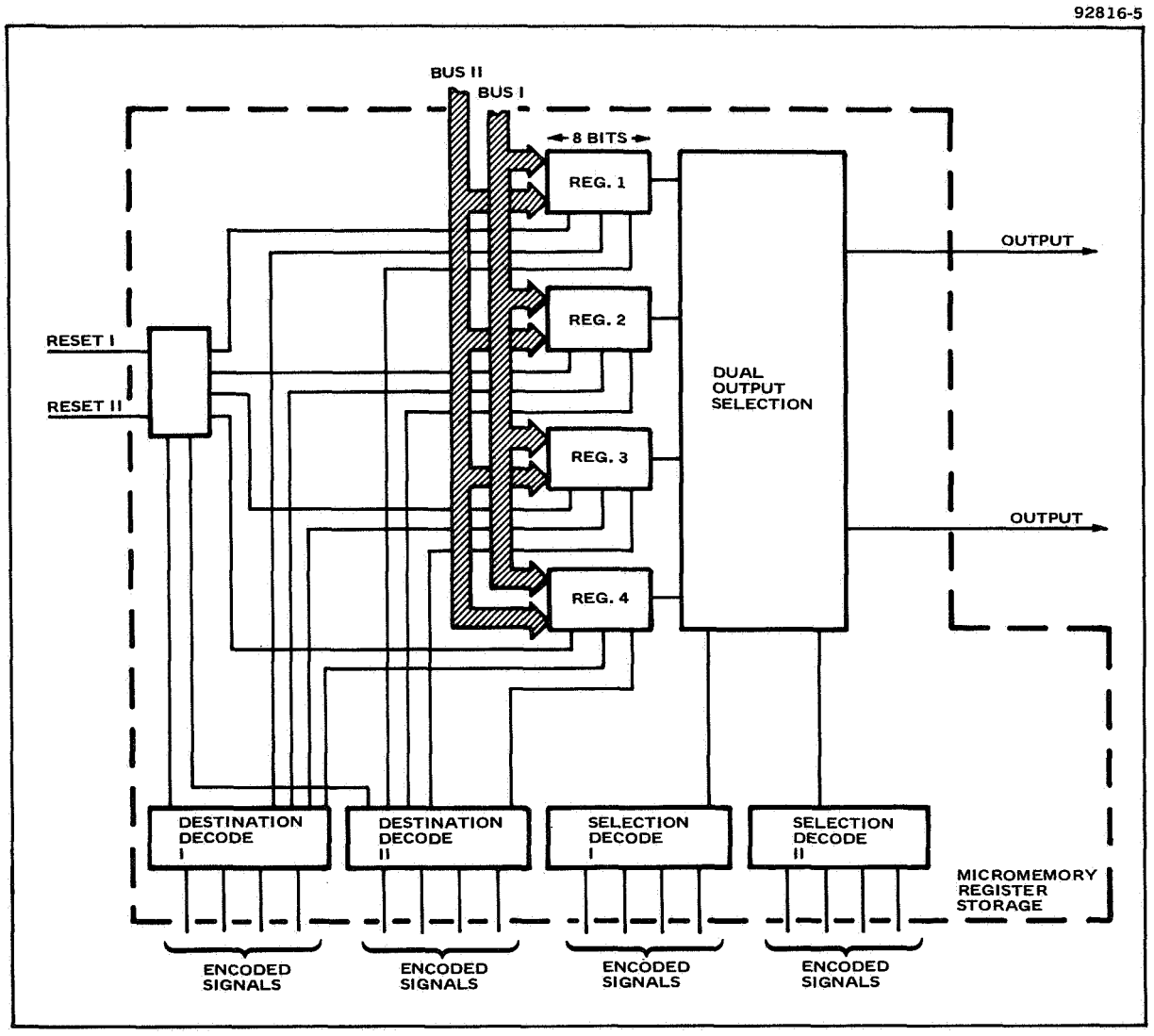


Figure 2-1. G1 Character Block Diagram. Operands of the microprogram are stored here.

DESCRIPTION OF THE L1 CHARACTER

The L1 character provides the basic logic functions in a centralized location at the heart of the microprogram controlled logic. The easy accessibility of these fast basic functions provide a means for efficiently building specialized complex operations.

The L1 character provides the basic logic functions selectable by the microprogram. The logic functions provided consist of rotates, shifts (logical), no-operation, complement, and incrementation. Also associated with the L1 character is the decoding logic for these logic operations. The type of microprogramming used with the functional character system relies heavily upon the fast and efficient manipulation of bits within the various operands. To this end, shifts and rotates have been provided which execute from 1 to 31 positions in a single step (as opposed to serial operation). Incrementation is accomplished with the use of a logic register which may also be used as a simple holding register. The L1 character is 8 bits wide and contains the following logic:

1. Bussing gates
2. Decoding logic
3. Rotate, shift, complement logic
4. Incrementer
5. L register
6. Gating to output bus

Figure 1-3 shows a block diagram of the L1 character. Several L1 characters may be connected to form logic operations on words longer than one byte. A limit of 4 bytes exists in order to maintain consistency of definition in the rotates and shifts.

Information entering the L1 card from the various sources is bussed to form the input. Then it is operated upon by the function selected in the current micromemory word and decoded on this character (left side of Figure 1-2). The resultant is bussed to the output where it leaves the character or is optionally stored in the L register (where it would thus be available at the next micro-instruction time for use in the increment operation or as an "L" source).

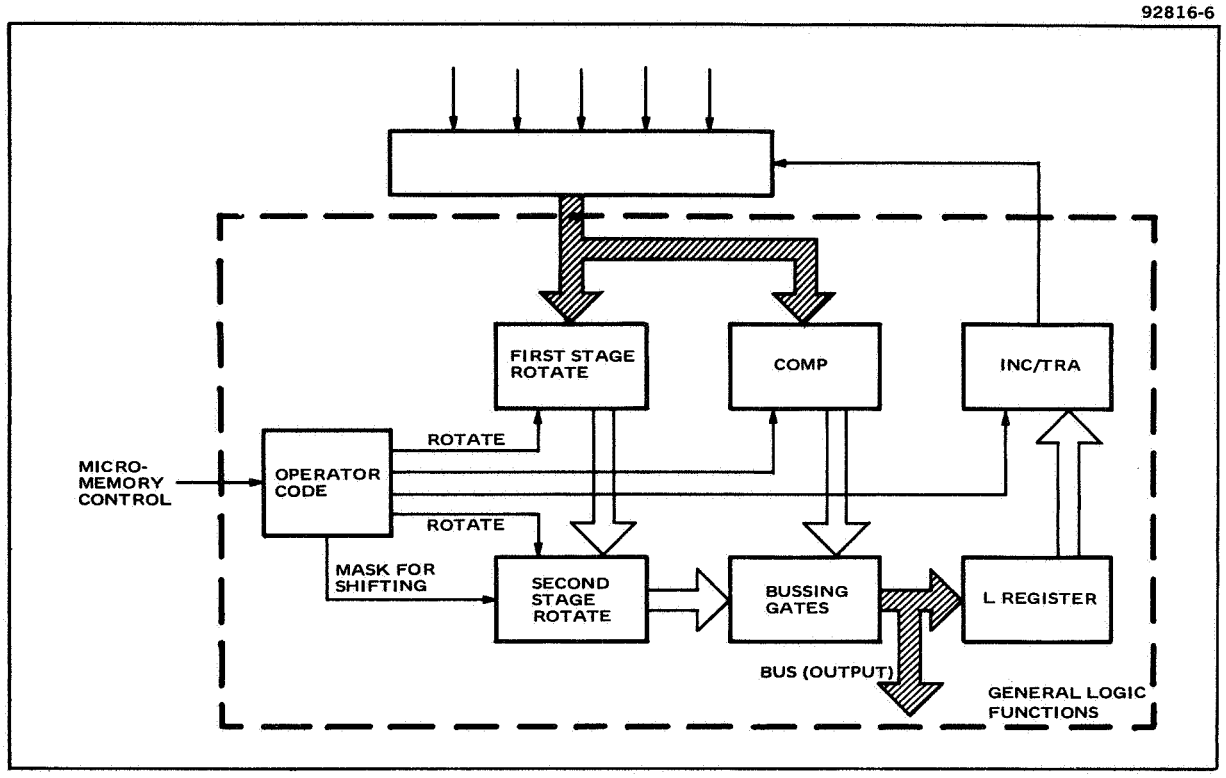


Figure 2-2. L1 Character Block Diagram. The shifts and rotates provided execute from 1 to 31 positions in a single step, for fast and efficient manipulation of bits within the operands.

DESCRIPTION OF THE L2 AND L3 CHARACTERS

The L2 character provides the microprogram's major arithmetic functions while the L3 character provides input/output logic capability. These major logic functions are thus provided as optional since in many situations they are not required.

L2 Character — The L2 character provides the major arithmetic functions for use by the microprogram. Addition is performed with carry look-ahead byte parallel. Control signals may condition the adder to alternately provide either of two special results (a) a mod 2 addition instead of full addition or (b) an input carry to the lowest order bit for subtraction. All arithmetic is in 2's complement form. The L2 character consists of two holding registers for the operands of the adder, the adder itself, decoding and error logic, and bussing gates. Figure 2-3 diagrams functionally the L2 character.

A typical arithmetic operation using the L2 character might proceed as follows: (1) first operand transferred to B register, (2) second operand transferred to A register (3) after appropriate delay access result and transfer out of L2 character. The error logic provides overflow and carry-out information.

L3 Character — The L3 character provides input/output capability for the microprogram machine. The L3 character provides input gating for external devices — four storing and three nonstoring channels. The storing input gating may be controlled by either the microprogram or the external I/O device itself. Four I/O output channels are provided. Interrupt signal storage and interrupt mask storage for four channels are available. Parity generation and check are provided for the four storing channels. L3 also contains the necessary destination and selection logic. Figure 2-4 is a block diagram L3.

To input data, an input line is selected under microprogram control resulting in selected data entering an E register, or in the case of a nonstoring input, entering the character output select logic. To output data, the micromemory places the data in the appropriate E register.

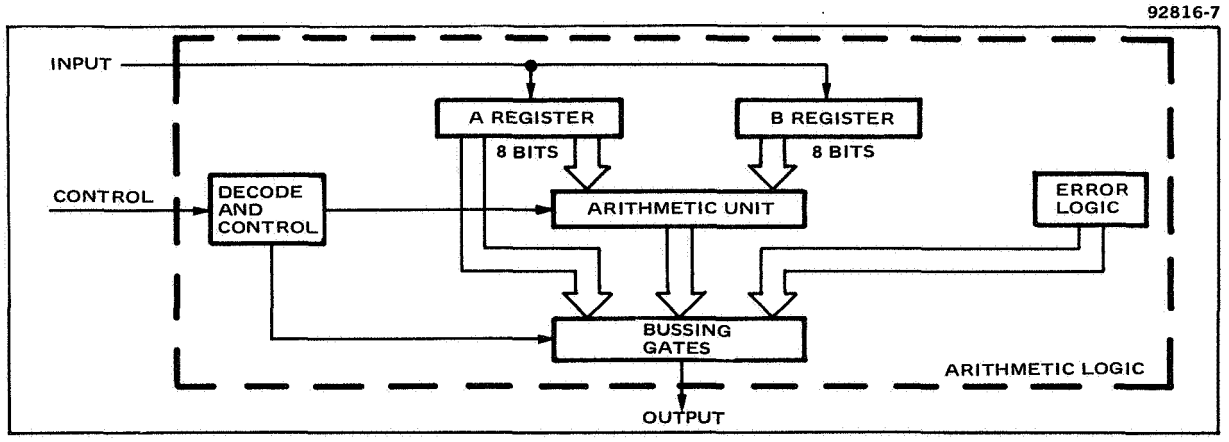


Figure 2-3. L2 Character Block Diagram. The L2 character performs all arithmetic in 2's complement form and addition with carry look-ahead byte parallel.

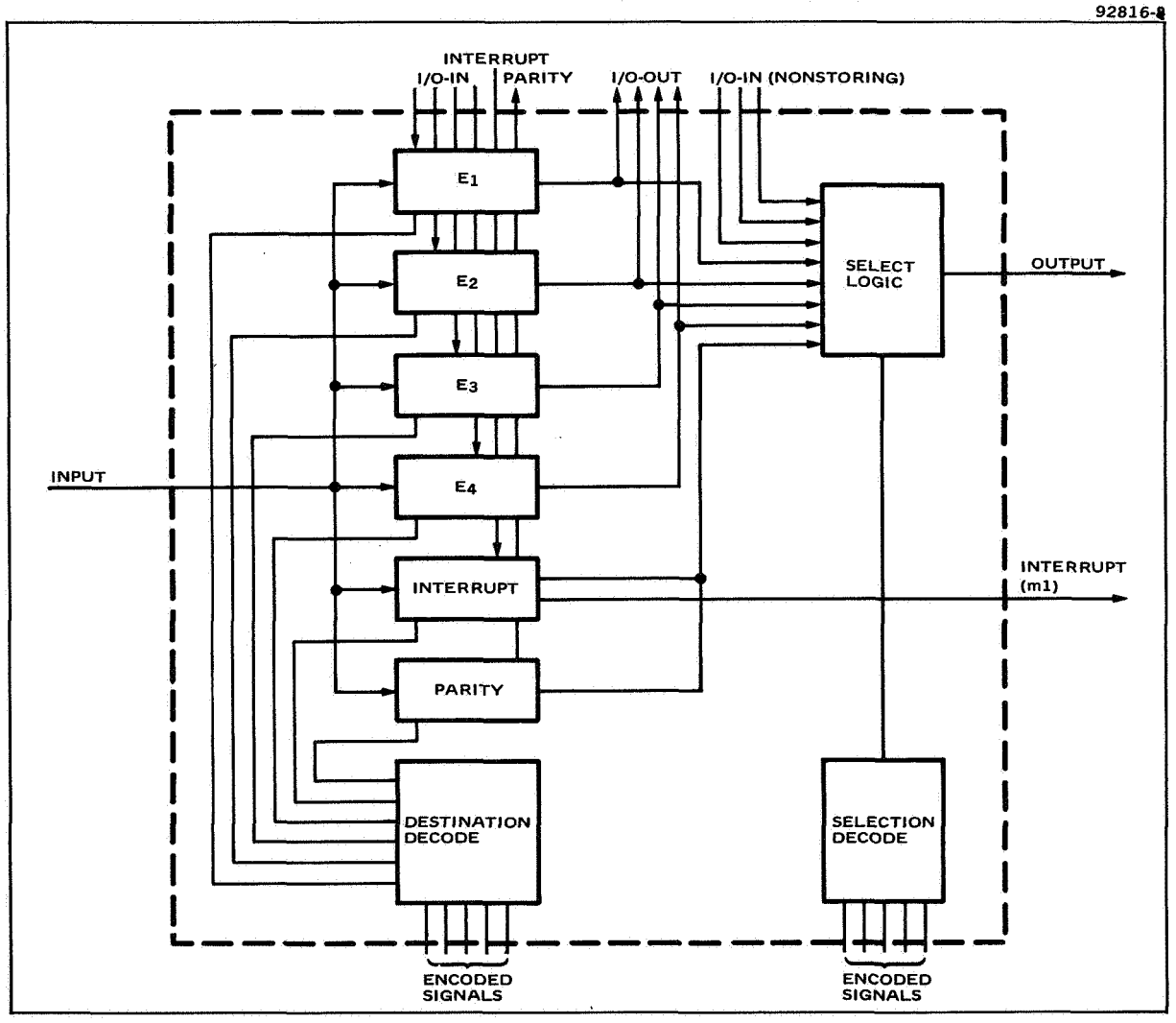


Figure 2-4. L3 Character Block Diagram. The microprogram I/O capability consists of eleven I/O channels, interrupt signal storage and interrupt mask storage, parity generation and check, and destination and selection logic.

DESCRIPTION OF THE MICROMEMORY CHARACTERS

The micromemory is partitioned into three characters to provide maximum flexibility. These characters provide the functions of the micromemory addressing, the micromemory array, and a micromemory word register.

The M1 character, micromemory address register, allows for addressing up to 1024 micromemory words; the M2 character, micromemory word register, provides for a full word; the MM character is a read-only micromemory array consisting of 256 words.

M1 Character — The M1 character provides the micromemory address register and related functions. The 10 address bits of M1 allow for addressing up to 1024 micromemory words. The address is contained in the MMC (Micro Memory Counter) register and serves to address the micromemory proper. The MMC register contains the address of the next micro-instruction word to be accessed. Associated with the MMC register is an incrementer which automatically steps through microprogram address states. This steps the microprogram sequentially until the program issues an unconditional transfer command. This transfer command takes the program out of the present sequence. Figure 2-6 shows the block diagram for M1.

Branching or transferring within the microprogram is provided by two modes: Unconditional transfer and conditional transfer. The M1 character is unique to a generating system and therefore appropriately carries the time base whose signal is distributed to other characters.

M2 Character — The M2 character contains a micromemory word register. The register provides for a full micromemory word. The micromemory word is divided into three fields. The first and the second fields are instructions, and the third is a constant. The second instruction is transferred into the register location of the first for execution resulting in sequential execution of the two instructions in the micromemory word. Timing is derived from the timing base. Figure 2-7 shows the block diagram of M2.

MM Character — The MM character contains the micromemory array. The address register and word register for the array are located on M1 and M2 respectively. MM is a read-only array. The presence of an address on the input lines causes the contents of referenced location to appear on the output lines after an appropriate delay and to remain there until the input address is removed. The MM character consists of 256 words of 50 bits each. Figure 2-5 shows the block diagram of MM.

Several MM characters can be combined to form a larger micromemory array. The maximum organization is 1024 words.

92816-9

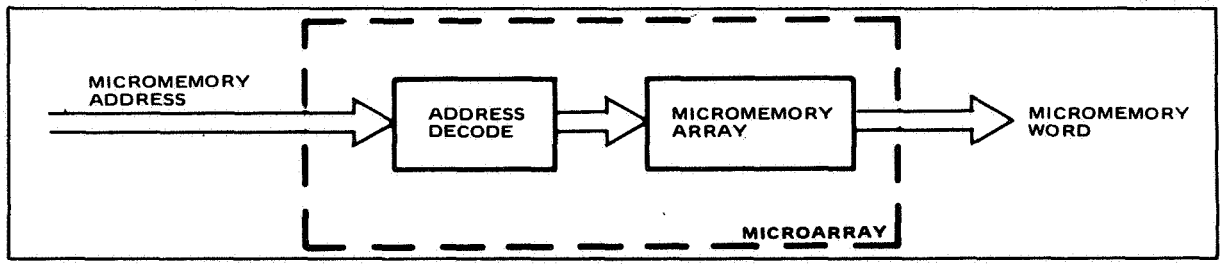


Figure 2-5. MM Character Block Diagram. Several of these arrays can be combined to form a larger micromemory array, to a maximum of 1024 words.

92816-10

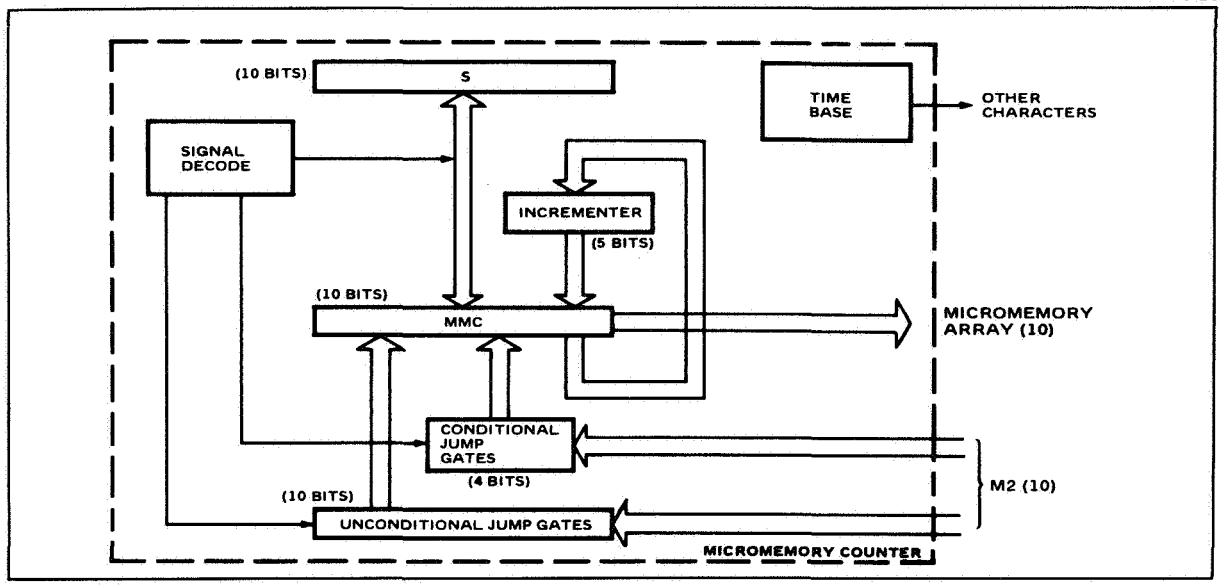


Figure 2-6. M1 Character Block Diagram. Ten address bits are available to address up to 1024 micromemory words.

92816-11

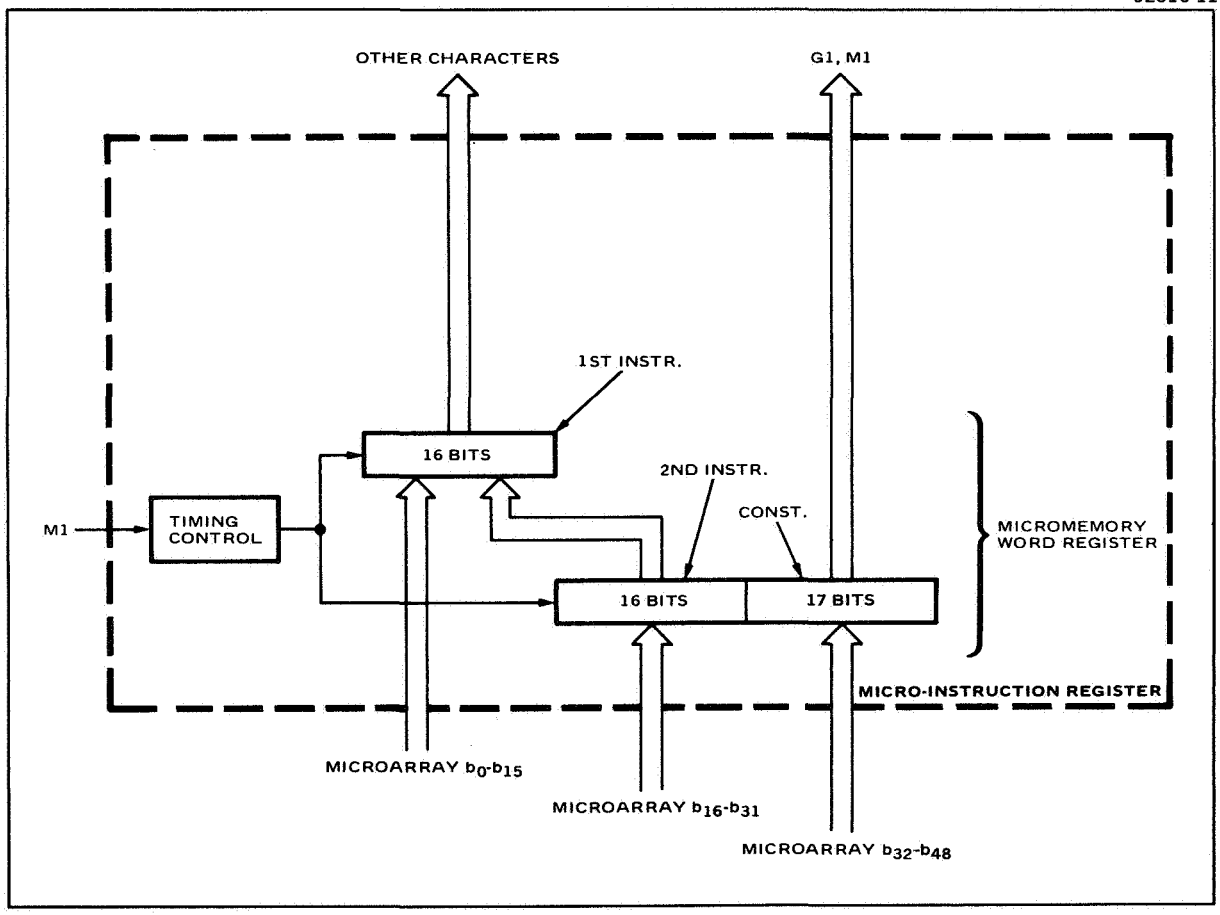


Figure 2-7. M2 Character Block Diagram. The register holds a full micromemory word.

DESCRIPTION OF THE P1, P2, and P3 CHARACTERS

The characters P1, P2, and P3 provide the secondary functions of scratch pad memory, up/down counting, and switching. These characters are one level removed from the main data loop since their functions are required less often than those of L1, L2, and L3.

The P1 character provides 256 bits of storage in a scratch pad memory; P2 is an expandable 8 bit up/down counter with byte look-ahead logic; and P3 is a switch allowing any three input channels to be switched to any three output channels.

P1 Character — The P1 character is a scratch pad memory of 256 bits of storage. The scratch pad is arranged into 16 registers of 16 bits each. Figure 2-8 is a block diagram of P1. The P1 character is connected to the L3 character through which its data flows. Up to 16 P1's may be connected in series to produce a total scratch pad of 256 registers. Generally the bit width will match that of the logic unit.

P2 Character — The P2 character is an expandable 8-bit up/down counter with byte look-ahead logic. The introduction of a time signal produces a real-time binary clock. The counter can be read in parallel and is resettable to any desired value. Zero detection is provided which may optionally interrupt the microprogram and/or the main program. The P2 character is connected to the L3 character through which data and control pass. Figure 2-9 shows the block diagram detail.

The P2 character contains control logic allowing the counter to be in a run/stop state dependent upon microprogram control.

P3 Character — The P3 character provides the capability of switching any three input channels to any three output channels. A 16-bit width is provided. This configuration allows three simplex simultaneous connections. Figure 2-10 shows the block diagram for the switch.

The input and output channels of P3 may be connected to any external interfaces which are electrically compatible. Storage is provided on the character for 9 bits of control information establishing the state of the switch.

There is no restriction on the switch state; all possible configurations are allowed (such as 3 inputs to 3 outputs, 1 input to 3 outputs, 3 inputs to one output, etc.).

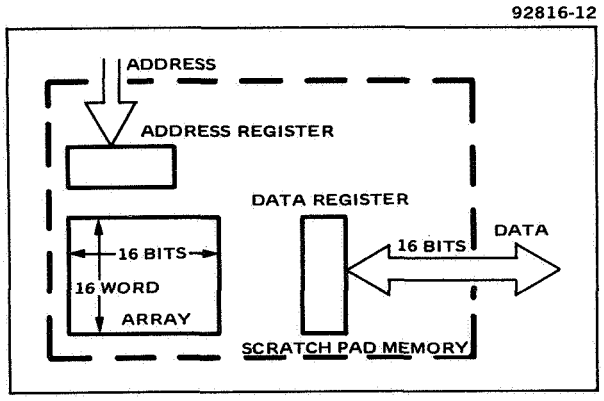


Figure 2-8. P1 Character Block Diagram. The 256 bits of storage are provided by 16 registers of 16 bits each.

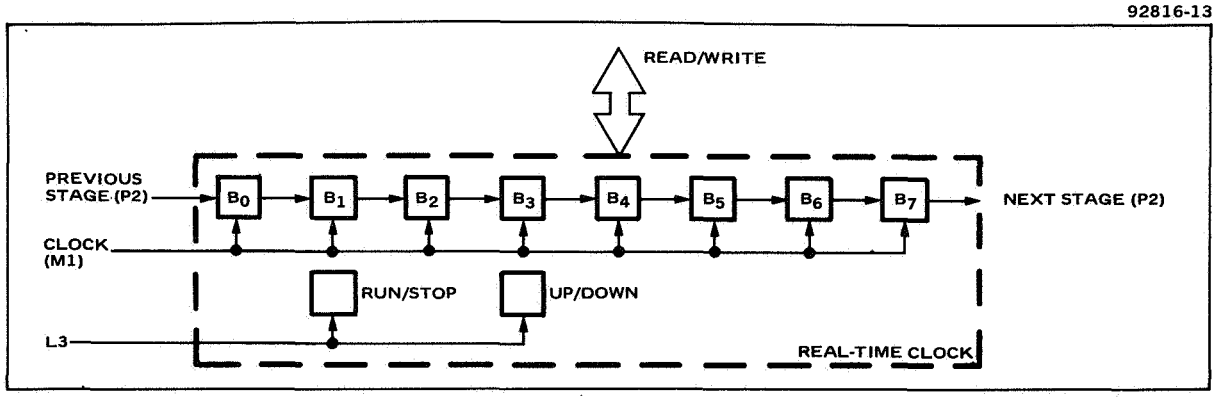


Figure 2-9. P2 Character Block Diagram. Introducing a time signal to this 8-bit up/down counter produces a real-time binary clock.

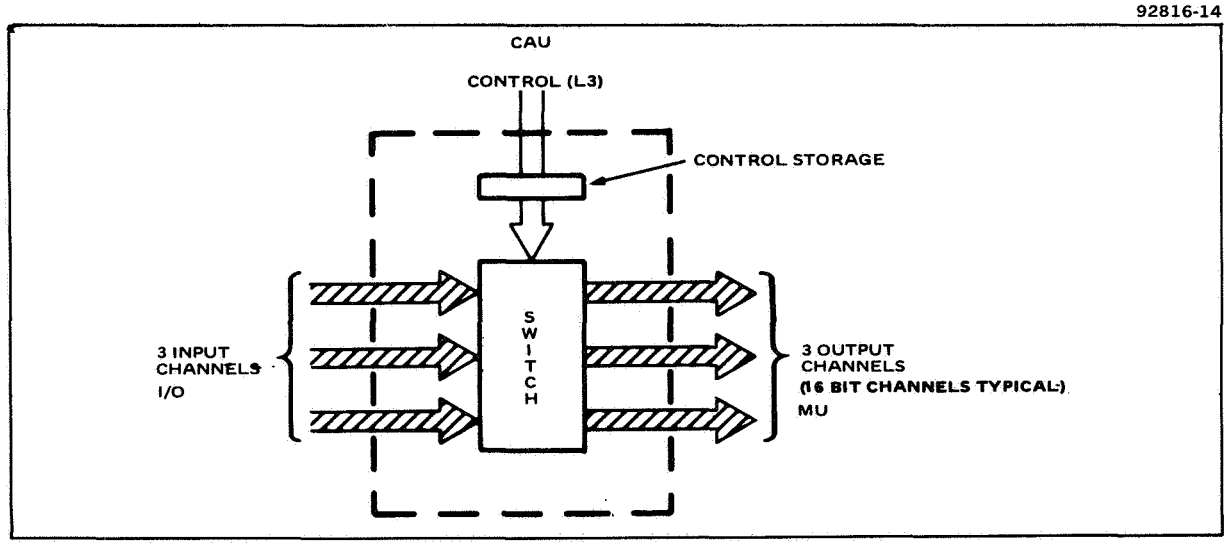


Figure 2-10. P3 Character Block Diagram. This switch allows three simplex simultaneous connections.

NECESSITY FOR THE USE OF MICROMEMORY CONTROL

Micromemory control allows modularization of control functions and straightforward design processes.

A universal conclusion among LSI researchers is that control functions are more difficult to modularize than functions related to data operations. Micromemory control technique was chosen as the solution for LSI implementation for several reasons. A micromemory, meaning here a read-only solid-state memory with its sequencer and instruction register, is easily partitioned into the large modules necessary for LSI implementation. Control functions in this form are then amenable to reproduction in large quantities of identical units. Also, design with control centered in one level of micromemory is more orderly and straightforward.

The micromemory has been provided with a relatively sophisticated microprogram instruction repertoire. This means that the microprogram contains the essence of the machine's major mathematical functions, such as multiply and complex sequencing. This is desirable since it represents an efficient use of hardware for these purposes and also reduces the number of different array types necessary. Also, a versatile repertoire leaves the designer free to make units whose operation is as simple or as complex as desired. The degree of flexibility which this repertoire gives the character set is a major factor in its success. It should be stressed that the "micro operations" of the character set are as important a factor as its logic design. This fact, a critical one in all LSI solutions committed to micromemory control, cannot be overemphasized.

The micromemory word provides the control necessary for the functions of the characters under its direct influence. All these characters so controlled are defined to belong to a common instruction group. There is one and only one M2 character per instruction group. A phase group consists of usually one or two co-instruction groups containing a common timing base. There is one and only one M1 character per phase group as illustrated in the figure.

In a phase group containing two instruction groups,one micromemory word, accessed from the first micromemory array (MM), operates upon and through its logic unit while the other word, accessed from the second micromemory array (MM), operates upon a second logic unit. Operations are carried out simultaneously in each unit with some cross translation. The option of including a second micromemory word allows for greater system capability by providing simultaneous operations; however, this does not affect the number of bits in the data word. (The data width is independently variable by byte.)

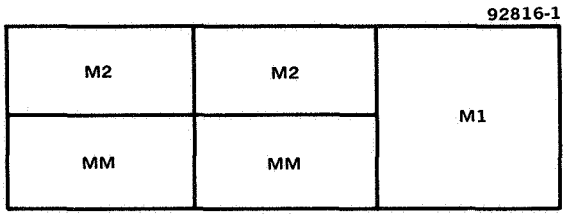


Figure 2-11. Micromemory Organization. There is one M2 character per instruction group and one M1 character per phase group.

FUNCTION AND COMPOSITION OF THE MICROMEMORY WORD

The micromemory word is designed with general instruction fields to provide a viable command capability for divergent programming requirements.

To fully utilize the building blocks of the previous section to implement the design of a digital system, a suitable microprogram repertoire of instructions is required. The repertoire is related to the scope of the logical functions to be performed and the amount of direction that can be conveyed to these logical functions within the contents of one micromemory word.

The micromemory word provides the control necessary for the functions of the characters under its direct influence. All these characters so controlled are defined to belong to a common instruction group. There is one and only one M2 character per instruction group. A phase group consists of usually one or two instruction groups containing a common timing base. There is one and only one M1 character per phase group.

In a phase group containing two instruction groups one micromemory word operates upon and through its logic unit while the other word operates upon a second logic unit. Operations are carried out simultaneously in each unit with some cross translation. The option of including a second micromemory word allows for greater system capability by providing simultaneous operations; however, this does not affect the number of bits in the data word. (The data width is independently variable by byte.)

A micromemory word is composed of two 16-bit fields, a 17-bit field and a one bit field — two instructions fields, a constant field, and parity bit. (See Figure 2-12.) The first and second instruction fields are identical, with execution of the second instruction following the first by half a cycle time (a cycle time is the time required for a complete cycle of the micromemory). The instructions can access the constant field, introducing into the data stream this constant from the micromemory. At those times when the constant field is not used as such, it takes on additional capability as a transfer field.

The instruction fields are each composed of three subfields — source, operator and destination as shown in Figure 2-13. The use of these subfields is described in the following topic.

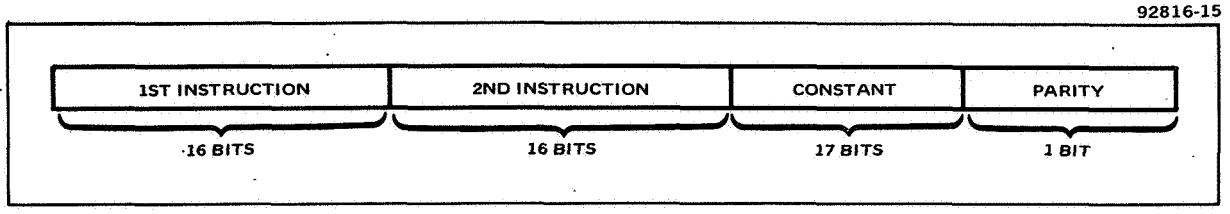


Figure 2-12. Micromemory Word. This word provides the control necessary for the functions of the characters.

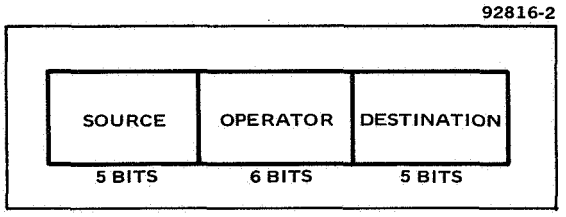


Figure 2-13. Instruction Subfields. The three subfields specify location of operand, operation to be performed, and destination of result.

THE THREE SUBFIELDS CONSTITUTING THE INSTRUCTION FIELD

Source, operator, and destination subfields compose the instruction fields of the micromemory word. The source specifies the origin of the data to be operated upon as defined by the operator field; the destination specifies the location where the data result will be stored after the operation is performed.

Each instruction field is divided into three subfields — source, operator, and destination subfields — which are described below in detail.

SOURCE SUBFIELDS — Each of the source subfields specify the source of information for the micro-command. The information selected by the source subfield is operated upon as specified by the operator subfield. Data accessed by the source code appears on the input data bus. The sources are described below.

G1 — G16 — The general set of registers providing the bulk of the fastest access storage in the functional system. A particular functional system may have 4, 8, 12, or 16 registers in a standard register unit. Register lengths may vary subject to the following constraints: (1) all 4 registers within one subgroup must be of equal length; (2) register lengths vary by an integral number of bytes. There are no restrictions on the use of the general registers except for the undefined state caused by setting and accessing a register within the same instruction.

E1 - E12 — These registers are part of the L3 characters serving as I/O buffers. These four registers per L3 character have the same characteristics as the G registers. In addition, the E registers provide for input and output channels to outside units.

CNT — The Constant Source allows for bits of the micromemory word to be accessed. These bits enter the input data bus right-justified.

INC — The Increment Source accesses the incremented value of the content of the logic register of the L1 character. The incrementer logic is unclocked so that after an appropriate delay the increment output is valid without further signals and remains valid and available for access until the L register content is changed.

A — The A Register Source accesses the content of the A register (a holding register for the Arithmetic Unit and part of the L2 card).

ADD — The Add Source accesses the output of the Arithmetic Unit. Usually this output is the full sum of the contents of the A and B registers; however, the output may be modified. These modifications allow for a forced carry into the lower position of the adder or inhibiting of all carries so as to produce an exclusive OR result.

L — The L register source accesses the content of the L register of the L1 character. Accessing and setting the L register within the same instruction time produces an undefined state in L; this includes the use of the INC source.

ECS — This transfers the error code onto the input bus of the logic unit.

*ADD — This source accesses the output of the Arithmetic Unit of a co-instruction group logic unit. (Two instruction groups are co-subgroups if they belong to a common phase group.) All conditions applying to adder access under ADD apply equally here.

OPERATOR SUBFIELDS — Each of the operator subfields specifies the type of operation the micro-instruction involves. These operators are listed below and operate upon the data from the input bus and present the result at the output of the L1 character. Each operator is itemized below along with a description of its function.

RS1 - 31 — This operator provides for a Right Shift of the operand from 1 to 31 position.

LS1 - 31 — This operator provides for a Left Shift of the operand from 1 to 31 positions.

MSK — The mask operator causes the source data to be masked by bits of the micromemory word. The bits of the source data are masked by corresponding bits of the mask. A mask bit of one and a source bit of one in corresponding positions will produce a one; all other combinations result in a zero.

NOP — The no-operation operator provides for data on the input bus to appear on the output bus without alteration.

R1 - R31 — This operator provides for a left rotate of the operand from 1 to 31 positions.

COM — The Complement Operator produces the ones complement of the operand.

DESTINATION SUBFIELDS — Each of the destination subfields specifies directly the register to receive the instruction result. These register designations are listed below:

G1 - G16 — This destination provides entry into the general set of registers providing the bulk of fast access storage for the functional system. These are the same registers described under the Source Subfield heading.

E1 - E12 — This destination provides entry into I/O registers of the L3 characters. These are the same registers described under the Source Subfield heading.

B — The B Register Destination allows the instruction results to be set into the B register of the L2 card. The B register holds one of the operands for the Arithmetic Unit.

L — The L Register destination allows the instruction results to be set into the logic register of the L1 card.

A — The A Register Destination allows the instruction results to be set into the A register of the L2 card. The A register holds one of the operands for the Arithmetic Unit.

*A — The *A Register Destination allows the instruction results to be set into the A register of a co-instruction group L2 character.

THE CONSTANT FIELD USED AS A TRANSFER FIELD

When the 17-bit constant field is used as a transfer field, it permits the microprogram to specify 1 unconditional and 11 conditional transfers.

The transfer field allows for microprogram specification of both conditional and unconditional transfers within the microprogram. The unconditional transfer provides a 10-bit address, the full microprogram addressing capability.

At all times when a transfer is not effected (either conditional or unconditional) the micromemory counter is incremented by one.

There are basically three testable functions: (1) least significant bit — true; (2) most significant bit — true, and (3) all bits — false (true = 1, false = 0).

There exist eleven conditional transfer test combinations and one unconditional transfer. The mnemonics used are defined as L = least significant data bit true; M = most significant data bit true; Z = all data bits false; I = data tested at input to logic unit; 0 = data tested at output of logic unit; and TRA = unconditional transfer.

DEFINITIONS OF TRANSFER MNEMONIC COMBINATIONS

LI	Tests the least significant bit for the "one" state on the input bus to the logic unit
LO	Tests the least significant bit for the "one" state on the output bus from the logic unit
LIO	Tests the least significant bit for the "one" state both on the input and output bus
MI	Tests the most significant bit for the "one" state on the input bus
MO	Tests the most significant bit for the "one" state on the output bus
MIO	Tests the most significant bit for the "one" state both on the input and output bus
LIMI	Tests the least significant bit and the most significant bit for the "one" state on the input bus
LIMO	Tests the least significant bit for the "one" state on the input bus and the most significant bit on the output bus
ZO	Tests all bits on the output bus for zero
ZOLI	Tests all bits on the output bus for zero and the least significant bit on the input bus for the "one" state
ZOMI	Tests all bits on the output bus for zero and the most significant bit on the input bus for the "one" state
TRA	Bits of the micromemory word are transferred to MMC, causing a micro-program jump; 10 bits from the micromemory instruction register are transferred to MMC

DESIGN COMPLEXITY ALTERNATIVES WITH FUNCTIONAL CHARACTERS

Simply by varying the number of characters and microprograms the designer may vary word length and functional capabilities of a single processor. He may also achieve parallel and multiple computation.

Figure 2-14 illustrates the levels of machine complexities available to the designer. Part A illustrates a very basic 8-bit machine, with simple logical, I/O, and register capabilities. Part B is the machine of part A expanded to 16 bits in its logic and register portions; however, no new functional capabilities have been added. Functional expansion is demonstrated in part C, where an 8-bit adder card and four 8-bit registers are added. Part D represents a significantly greater jump. Illustrated is the dual-logic unit capability of the character set. If desired, it is possible to have two logic units, with different but coordinated microprograms, operating in parallel. They share the same sequencer (M1), which both control. The G1 bank is common to both logic units.

Part E illustrates an even higher level of expansion. Two totally independent micromemory units (memory and sequencer) drive three different logic units, linked together through G1 cards. This level of complexity can be carried to an almost limitless expansion of micromemories and logic units bound together by shared G1 characters.

With the hardware specified, the next major task is the writing of microprograms. In machines of this type this is as important as the hardware design. Often the only essential difference between units designed for different purposes is their microprograms.

As an internal R&D program, Hughes built and tested a small expandable computer to demonstrate the feasibility of the Functional Character Method. The model consists of a truly expandable system as shown opposite. The characters used to build this computer contain 4 bits per character as opposed to 8 for the characters initially defined. They also have other simplifications, but do not lose any pertinent structures or characteristics.

This model machine was built and demonstrated as described by merely adding characters to step from stage to stage without making any changes in the back panel wiring or characters already in place, thus, demonstrating a truly modular capability for digital systems using the character set and its methodology. The number of degrees of freedom provides for minimum systems consisting of as few as five characters to large systems limited only by the organization of the microprograms and macroprograms. The number and type of characters determine the physical size and capability, while the microprogram that controls them produces the system characteristics.

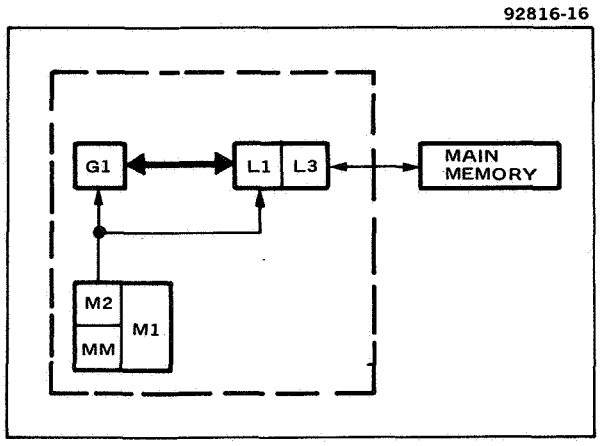


Figure 2-14A. Basic Machine

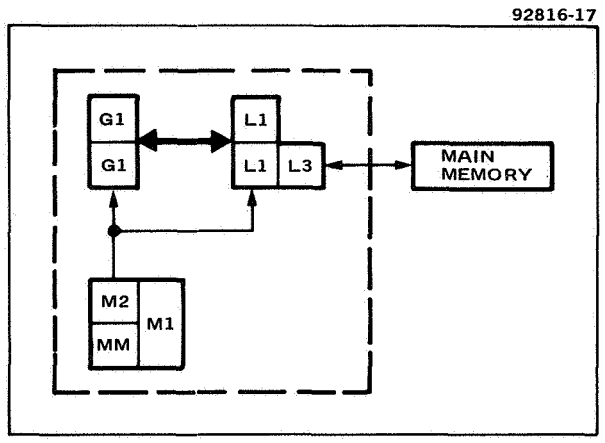


Figure 2-14B. Data Length Expansion

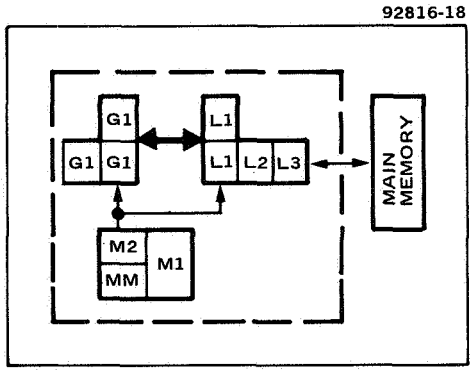


Figure 2-14C. Functional Expansion

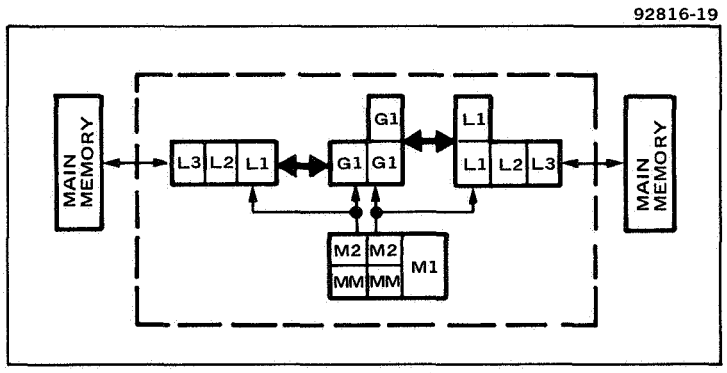


Figure 2-14D. Parallel Computation Expansion

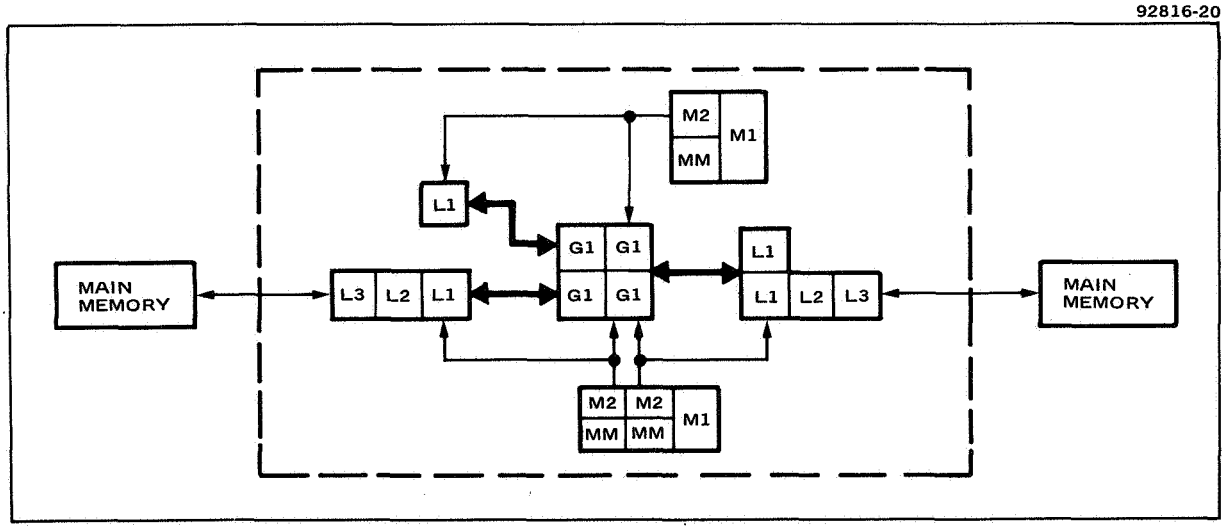


Figure 2-14E. Multicomputation Expansion

Figure 2-14. Four Stages of Expandability. A comparison of Parts A and E illustrates the versatility of the character set as it is adapted to both simple and complex situations.

RELIABILITY AND MAINTAINABILITY USING FUNCTIONAL CHARACTERS

Although design with the character set is high in gate usage, the characters impose no speed penalty and give significant advantages in maintainability, reliability, and diagnosability.

Design work to date indicates that most digital data processing equipment can be implemented using only the 10 characters. Gate counts run higher than equipment configured from discrete IC's, with 140% of the IC gate count representing an approximate upper bound. Speeds appear to be comparable to the latest airborne development computers, and promise to be competitive with ground equipment as well.

For all systems where maintainability is a factor, units constructed from the character set have the obvious advantage that only 10 types of spares are needed to insure system repairability. Nine of the characters are identical in all applications. The tenth, the micromemory, stores a unique program for each application. To bypass the requirement for spare ROM's of specific patterns, research is currently underway at Hughes to develop an electrically alterable ROM. It would be possible to deliver the MM characters "blank" from the manufacturer to be written into by the user with a one-shot process, and thus the memory would represent a single part to be spared.

Reliability of character-built LSI computers is enhanced by the reduction in the number of lead-bonds.

The characters enjoy a natural advantage in the diagnostic field. The arrays establish replaceable units which are quite large, thus minimizing the degree of fault isolation required. The bussed structure provides several convenient points for application and observation of diagnostic signals. Also, there are only a certain number of allowable ways to interconnect characters. This, plus the fact that there is no intervening logic, precludes the possibility of unexpected timing or logic problems arising. Once the fault detection and isolation problems are solved relative to a character, the solution is applicable to all combinations in which that character is found.

Furthermore, since every character is under the control of some micromemory, a third major approach, along with more traditional hardware and software approaches, to diagnostics becomes available. Investigations have shown that microprogram techniques are extremely effective in both detecting and isolating faults in the characters. This approach also promises fast diagnostic speeds. Not only are the diagnostics carried out at microinstruction speeds rather than machine-instruction speeds, but in large machines each micromemory can simultaneously diagnose the characters under its control.

As an example, consider the application of these techniques to the diagnosis of the NASA MCB. Each of nine micromemories can simultaneously diagnose 7 to 38 characters each. Any fault need be isolatable to one of only 206 characters, for which a replacement is chosen, assuming an operator is present, from 10 basic part types. (Of course, the NASA MCB actually reconfigures automatically in case of error.)

Problems currently under investigation are diagnosis of the micromemory itself, amount and type of hardware required, and the applicability of more conventional techniques. The table shows preliminary results as to the amount of hardware required for diagnosability for the various MCB units. The level of fault isolation for the MCB with this method is 89%. It may be possible to develop techniques for 100% fault detection and isolation to the character level.

These diagnostic results were developed under Hughes internal funding in a parallel effort in evaluating the characters.

TABLE 2-II. ADDED HARDWARE AND HARDCORE % FOR THE UNITS OF THE NASA MCB — DIAGNOSTIC MICROPROGRAM APPROACH

Unit	Added* Hardware %	Hardcore %
NASA MCB CU	16.2%	0.94%
NASA MCB AU	16.3%	0.88%
NASA MCB I/O	21.5%	1.09%
NASA MCB MU	35.8%	2.64%
NASA MCB CAU	7.8%	0.40%
NASA MCB Entire System	16.5%	0.95%

*Includes actual diagnostic hardware and all necessary ROM for storage of diagnostic microprograms.

Division of added hardware between logic and ROM space is about 45%:55%. The level of fault isolation for the MCB with this method is 89%.

Definition of Terms:

Added Hardware % — The percentage by which the hardware of the system increases to provide diagnostic capability.

Hardcore % — The percentage of hardware in which faults will be undetectable.

Level of Fault Isolation % — The percentage of faults which may be isolated to one character.

(These results accomplished as part of an overall evaluation of character diagnosability performed under Hughes Internal Development Funding.)

AIDS FOR DESIGN AUTOMATION CAPABILITY WITH THE CHARACTER SET

The character set provides the potential for a complete design automation process after the necessary software is developed.

The area of application stressed for the character set has been computer implementation. Though the computer makes a meaningful application, there is, however, great economical advantage to be gained through application of the characters to digital equipment of unique or low volume design. Using the character methodology in such systems can reduce by large factors the engineering costs, design, and checkout time involved. To effectively achieve such a goal several design aids are used--a character assembler, a microprogram assembler, and a system simulator. These three programs allow for complete design automation capability.

The character assembler input consists of encoded instructions having the information content of a block diagram as exemplified by Figure 2-15. This information in conjunction with the character characteristics (which form the data base of the assembler) is processed by the assembler to produce an output consisting of wiring information for the interconnection of the characters. The character assembler output may be in the form, for example, of a wire list, an N/C tape for automatic wiring machine, or a tape input to a routing program for printed circuit card etch layout.

The encoding information for the micromemory array is provided on tape by the microprogram assembler. This tape is used directly in the manufacture or alteration of the array. The microprogram code is assembled with the usual aids provided by machine language assemblers.

System simulation is accomplished from (1) information of the machine structure as input to the character assembler, (2) the microprogram code as input to the microprogram assembler and (3) instructions from the system designer input directly to the system simulator. The degree to which system checkout is dependent upon the sophistication of the simulator. However, because of the high level of definition of the characters themselves the simulator need not be concerned with details of the Boolean logic or signal interface consistency between characters. Therefore a worthwhile simulator is seen as an achievable goal.

Thus, the complete system--microprogrammable characters, character assembler, microprogram assembler, and system simulator--provide the system designer the capability for total system design from his desk. Furthermore, he is not concerned with logic design in any form. When he specifies the following:

1. character configuration
2. microprograms
3. simulation instruction

these items are provided for:

1. character assembly
2. back panel wiring

3. micro-array encoding
4. system checkout

all without the services of a logic designer or the technician's help. In fact, it is conceivable that no human intervention need take place between the system designer and his designed hardware.

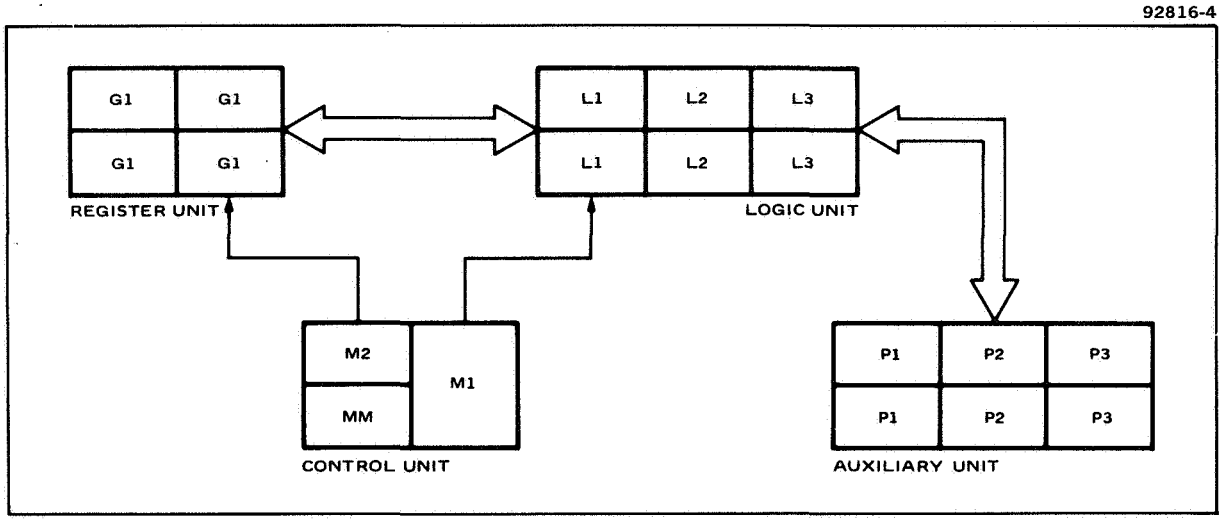


Figure 2-15. Typical Functional Character Configuration. Functional character configuration is a major input in the design automation process.

STEPS IN AN AUTOMATED DESIGN PROCESS USING FUNCTIONAL CHARACTERS

- Designer Identifies Information in Above Diagram (number of characters, type of characters, interconnections)
- Character Assembler Produces Wiring Data
- Designer Encodes Micromemory Array
- System is Simulated Based on Machine Structure and Microprogram
- Micromemory Arrays Manufactured from Array Encoding as Modified After Simulation
- System is Assembled and Checked-out

BLANK PAGE FOLLOWS 2-25/2-26

SECTION 3
MCB COMPUTER PRELIMINARY IMPLEMENTATION

Section 3 — MCB Computer Preliminary Implementation

INTRODUCTION TO THE NASA MODULAR COMPUTER

NASA and Hughes are exploring means of implementing a modular computer to achieve the long life and flexibility required for future space missions.

The NASA Electronics Research Center (ERC) in Cambridge, Massachusetts, has undertaken a broad program to satisfy flight computer system requirements for future missions, including versatility and long term reliability. Specific attention to these requirements is necessary because flight qualified aerospace computers, and even some still under development, have been designed for increased computational speed and arithmetic capability, but not for the long life reliability and application flexibility that will be required for future space missions. For example, the mean time between failure (MTBF) of available aerospace computers lies in the range of 2,000 to 5,000 hours, whereas long space missions will require an MTBF of 10^5 hours.

Several computer organizations have been described in the literature which include redundancy for increasing mission reliability, but still neglect applications versatility. Some non-spaceborne computers of the array or multiprocessor type are currently being developed. These systems, although potentially capable of meeting ERC's versatility and reliability objectives, lack design features for space applications (component reliability, weight, volume, radiation hardness, etc.).

To satisfy the requirements, NASA ERC is studying a modular computer organization. This Modular Computer, as a potential component of a guidance and navigation subsystem of several potential space booster configurations, must be applicable to at least four distinct missions: the synchronous satellite, lunar orbiter, Mars orbiter, and Jupiter fly-by solar probe. Computer memory size, word length, and speed requirements for each phase of these four missions have been estimated by means of computer simulations. The object computer was assumed to have single-address and sequential operation.

The most severe requirements in terms of speed and accuracy occur during boost. Post injection computational requirements are low and the accuracy of computations is far less critical. Therefore, to satisfy the composite requirements a Modular Computer (MC) organization as shown in Figure 3-1 has been structured. Each column of the MC can satisfy a 1.5×10^5 instructions/sec requirement.

The computer can be configured to operate as a number of parallel processors, with each segment or column solving an independent problem that may be different or identical. Each column in turn contains a number of blocks called modules, which may be configured so as to form patched columns, using modules from different physical locations; for example, a diagonal (see Figure 3-1). This structure meets the high speed computational requirements for attitude control associated with strapdown systems, and also achieves the reliability required for long time mission success.

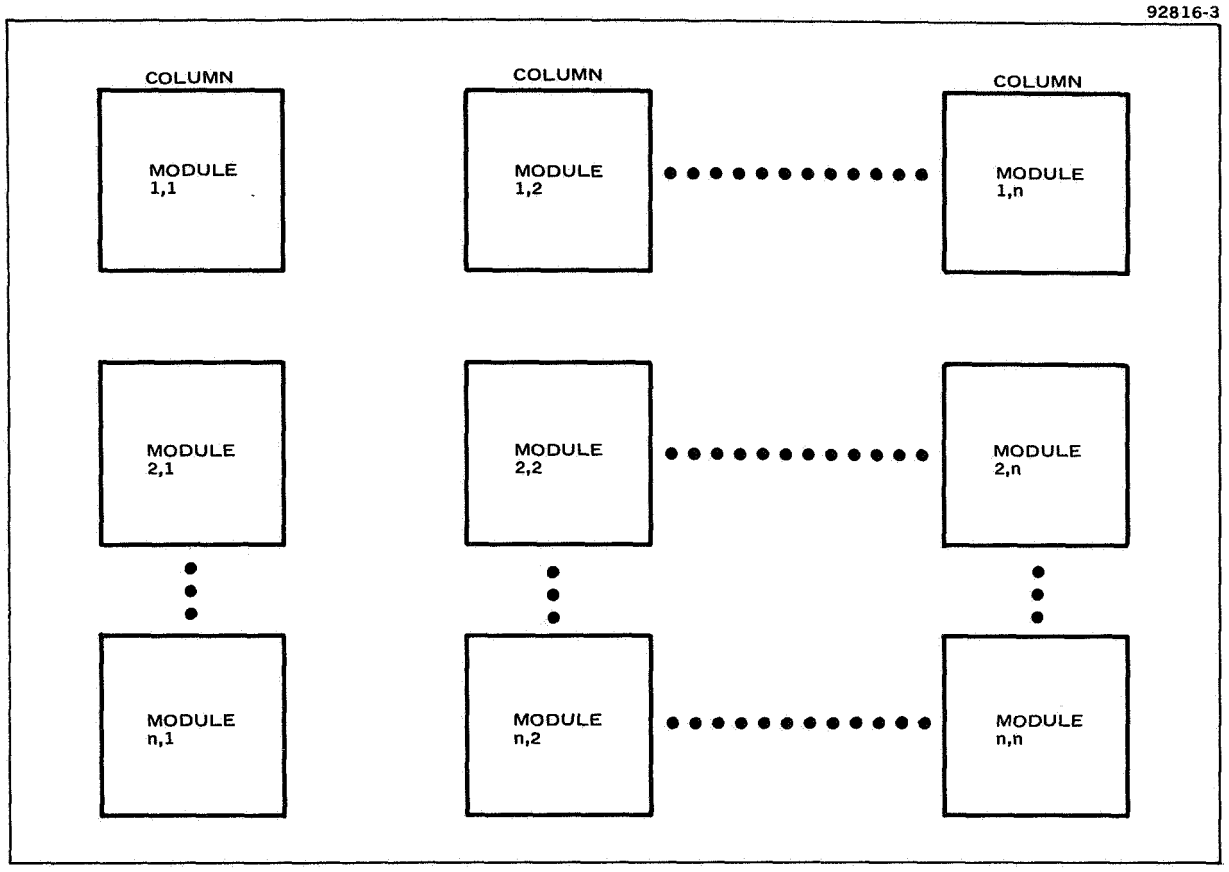


Figure 3-1. NASA Modular Computer Showing Columns and Modules. Identical modules in rows provide redundancy while working computers can be configured in many ways from the distinct columnar module types.

THE MODULAR COMPUTER BREADBOARD (MCB)

A breadboard version of a two-column subset of the modular computer has been constructed to investigate mission algorithms, parallel processing, and automatic reconfiguration.

A breadboard model of the modular computer consisting of two columns has been built and is currently in the terminal stage of system checkout. Software is being developed concurrently with hardware. This Modular Computer Breadboard (MCB) will be used for experimenting with different structures in order to enhance the NASA ERC modular computer objective. The organization of the Modular Computer in a three column configuration is shown in the figure.

During boost, the three columns of the modular computer operate concurrently in a triple modular redundant (TMR) mode, with majority voting at the outputs. After orbit injection, the TMR mode is terminated and the ensemble of modules is configured so that only one computer remains operating; the others are turned off to conserve power and improve reliability. (The failure rate of non-operating circuits is assumed to be lower than that for operating ones.) System interlocks are provided which insure that the on-computer performs correctly (within bounds). If this is not the case, the Configuration Assignment Unit (CAU) is triggered. It is the task of this unit to assemble at least one computer out of all the available modules.

The availability of good modules is determined by means of hardware-software tests with interlocks. As may be seen from the figure, each of the computers has been separated into four functional modules: a Memory Unit, Control Unit, Arithmetic Unit, and a IO Unit. The Configuration Assignment Unit, (CAU) in conjunction with the CU, together with the Configuration Control Switches (CCS), can automatically reconfigure the ensemble so as to form an operating computer. Such a computer may consist of any combination of MU-i, CU-i, AU-i, IO-i.

Although the figure shows a tri-column configuration, the actual flight computer may require additional columns and some configuration adjustment in order to meet the mission time requirements.

The breadboard version of the modular computer contains two columns. This is sufficient for the intended experiments:

1. Determination of mission algorithms within specified accuracy limitations and consistent with the intended application.

2. The use of parallel processing to achieve higher effective computational speed.

3. Automatic detection and isolation of the occurrence of a computer module failure, and automatic reconfiguration to eliminate the effects of the faulty element.

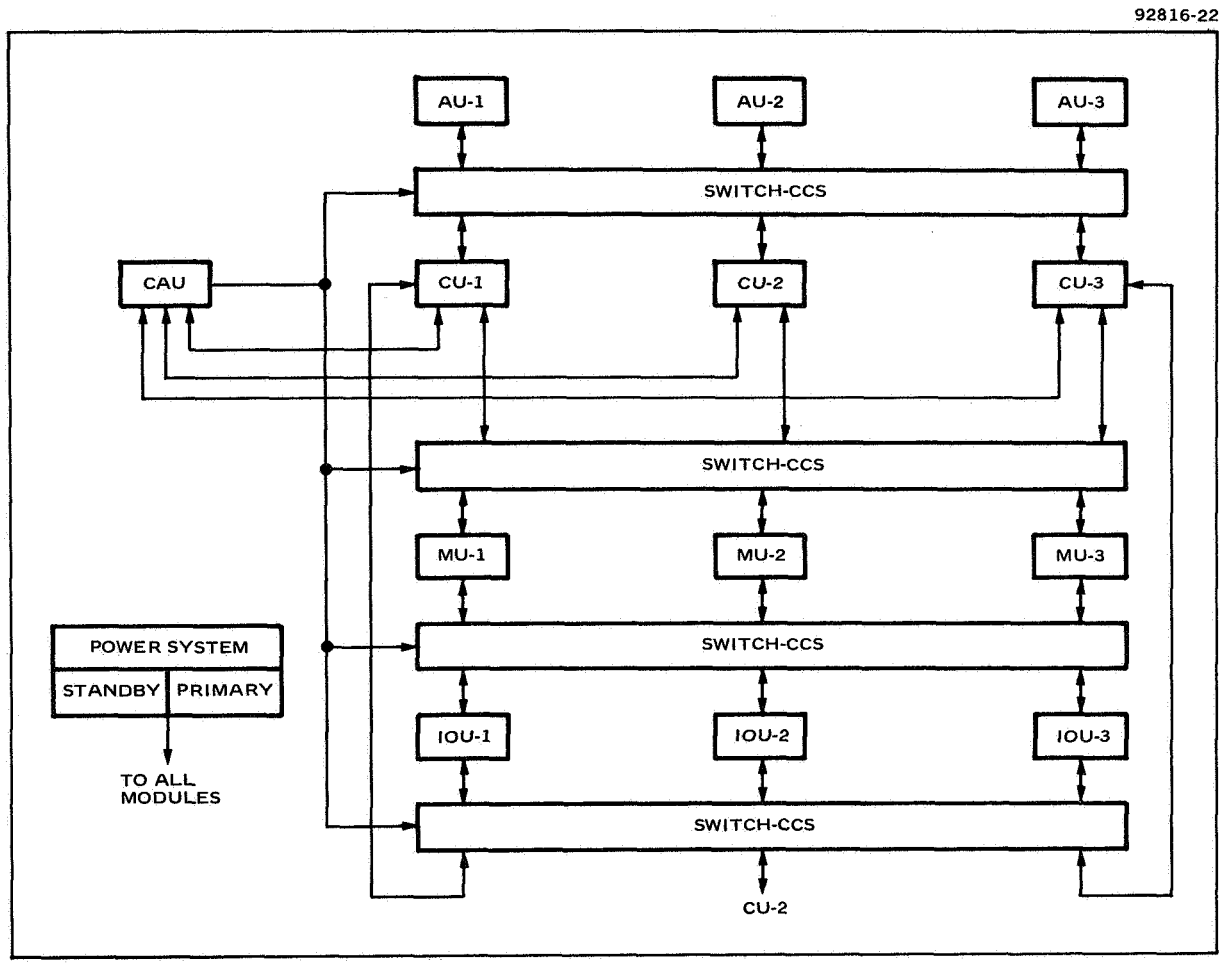


Figure 3-2. Modular Computer Organization. The breadboard is a two-column configuration which is sufficient to demonstrate computation, modularity, and automatic reconfiguration.

IMPLEMENTATION OF THE MCB WITH FUNCTIONAL CHARACTERS

Implementation of the MCB using functional characters demonstrated ease and speed of design together with the ability to retain required performance characteristics.

In order to evaluate the character set, the breadboard version of the modular computer (MCB) was implemented. Design with the character set is a straightforward procedure which may be described as follows:

1. Assign character types to the unit according to functional requirements.

2. Microprogram the unit in detail to accomplish its assigned function.

3. Determine the number of each type of character needed according to the requirements of the microprogram.

4. Revise the assignment of character types according to the needs of the microprogram.

Application of this procedure to the MCB revealed that the structure of each unit was similar. Each required L1 characters for storage of operands, and L3 characters for I/O functions and communication between units. The micromemory characters M1, M2 and MM were necessary in each unit for control. Further study of the functions of the units in the MCB revealed that L2 characters would be necessary only in the AU (for complex arithmetic) and CU (for effective address calculation). Assignment of P-characters was done strictly on the basis of need. Scratchpad memory requirements in the CU were satisfied by P1, counter requirements in the CU and CAU by P2, and switch requirements by P3.

Microprogramming was the major task of the design procedure. Although the structure of each unit was similar, the microprograms were quite different, reflecting the difference in the function of each unit. The microprograms determined the actual number of characters necessary and lead to revisions in the choice of character types. (For example, microprogramming revealed that G1 was not necessary in the MU.)

The initial design for the MCB was accomplished with about 6 man-months of effort, indicating the relative ease of designing with the character set. The CAU was the most difficult unit to design since it involved connecting characters (G1's) in other than common configurations. The MCB presented here is an improved, re-microprogrammed version compared to the MCB described in the first interim report.

The functional and operational aspects of each unit of the MCB have been preserved in the functional character implementation. This plus the similarity in performance specifications (such as hardware count and instruction execution rates) between the character set and original MCB verifies the ability of the character set to be applied successfully to a specific design problem. Some detail of implementation (such as control signals for inter-unit communication, use of memory arrays, etc.) was tailored to the character set, so this version should not be expected to be identical to the original MCB although it is functionally equivalent.

Figure 3-3 shows the character content of each module adjacent to the name of the module. The number to the left of the shash (/) is the total number of characters used per module. The number to the right of the slash is the number of character types used in that module. Note that the number of characters is additive, whereas the number of character types is not; the sum of the character types is 10.

The following topics describe in detail the character implementation of the MCB.

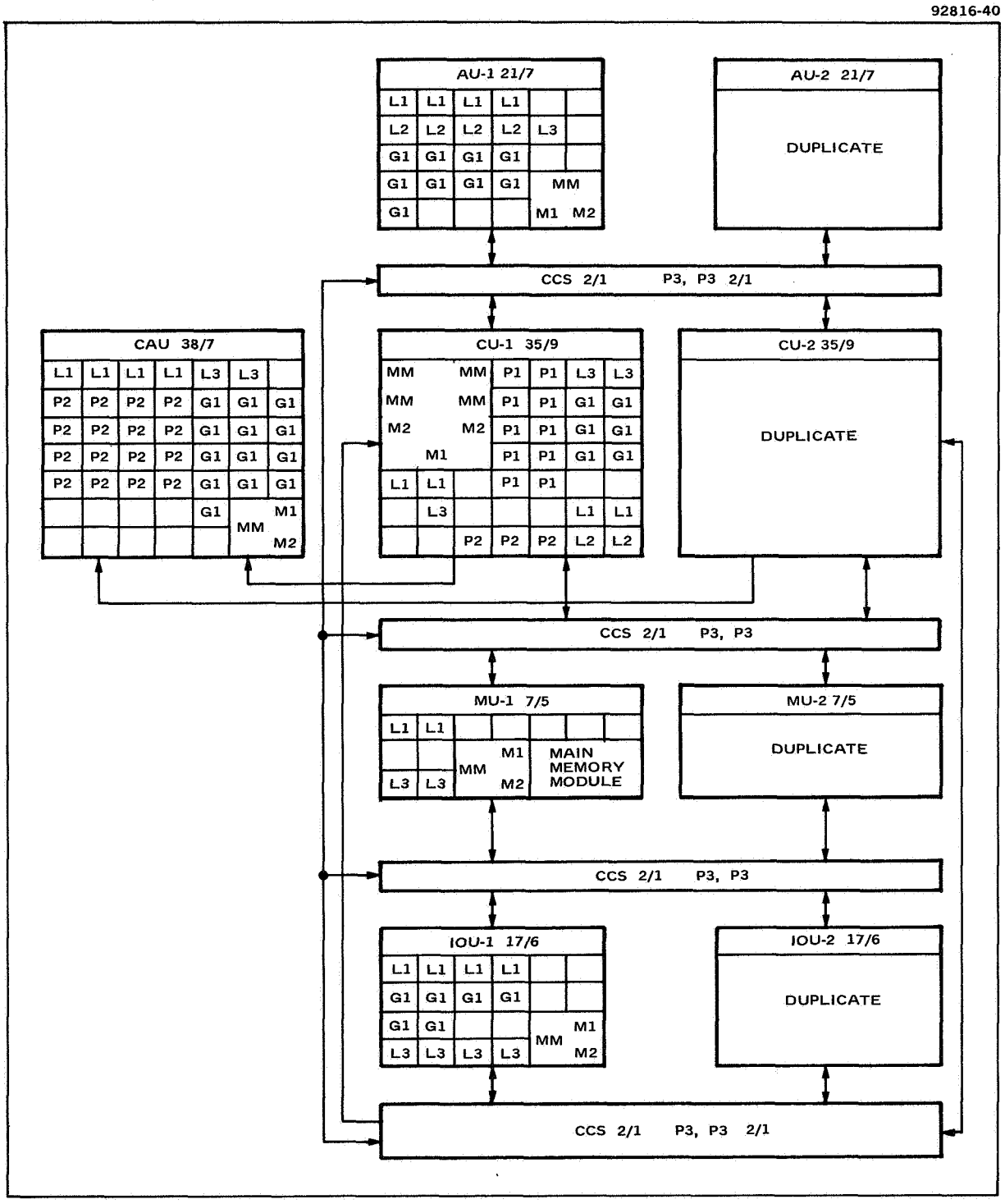


Figure 3-3. Functional Character Implementation of MCB. This implementation verifies the ability of the character set to be applied successfully to specific design problems.

OVERALL DESIGN OF THE MCB COMPUTER

Hughes has analyzed the MCB and prepared a computer design as a test of the generality of the functional character set. The Hughes interpretation of the MCB is presented in this and the following topic.

The existing design of the NASA MCB Computer has been emulated as a test of the generality of the functional character set. All aspects of the computer have been considered. Parity generating and checking will be done by hardware included as part of the L3 character. The contents of one register on each GI character will be available as direct outputs from the character. This, in addition to L3, will provide for inter-unit or external discrete signals. Because of the nature of the character set, internal error detection can be provided easily by a small diagnostic program in micromemory in addition to planned hardware error detectors. Work on the data block transfers has been completed.

Control Unit (CU) — The CU controls the operation of the MCB subcomputer of which it is part. Main program instructions are accessed and decoded; the CU then performs necessary effective address calculations and operand accessing. If the instruction is one that is performed in the Arithmetic Unit (AU), the CU transfers the instruction's operator field and both operands to the AU. If the instruction is an input-output data block transfer, the CU transfers the instruction to the Input/Output Unit (I/O). All other instructions are performed internally by the CU. If the instruction is one that produces an answer, the CU will store the answer in the specified location. Operation of the CU is described in more detail in the following topic.

Arithmetic Unit — The Arithmetic Unit receives the current instruction operator field and the actual operands from the Control Unit. It decodes the operator field, performs the operation accordingly, and transfers the resultant back to the Control Unit. The AU has the responsibility to perform all floating point operations, multiplies, and divides. The AU has the capability to detect overflow from arithmetic operations.

Memory Unit — Each MU contains one main memory module capable of storing 4,096 words. The address and data registers associated with the module will be read-write registers addressable by the MU microprogram similar to the G registers. The data to and from these registers follows the usual data paths of the characters. The MU interfaces with the Control Unit and Input/Output Unit. It can transmit data to and from either unit, acting on commands from either. The address must be provided by the CU or I/O. For input-output transfers, data flows directly between the MU and I/O units. The MU will respond first to the I/O in case of conflict.

Input/Output Unit — The Input/Output Unit is provided with four input and two output channels with expansion possible. It provides a data path between any of its external devices and the Memory Unit (for data flow in either direction) upon receipt of the proper commands from the Control Unit. An independent unit, the I/O is capable of controlling the MU and all input/output devices, and carrying out an entire block transfer. The interface with external devices is 32 bits wide. The CU maintains control over the input/output transfer to the extent that it can query the I/O as to status of the transfer and can, between words, overwrite one block transfer command with another.

Configuration Assignment Unit — The CAU continually monitors the idle-time counter and the diagnostic-time counter. If either is permitted by the CU to go to zero, the CAU immediately takes over the reconfiguration duties. After configuration, the CAU

notifies the appropriate CU by means of an interrupt that it is to perform a self-check diagnostic. If this fails, the CAU tries another configuration. The CAU also has the duty of comparing mask and status registers. If a corresponding "one" is detected, it notifies the appropriate CU to start a diagnostic which may eventually arrive at a new configuration. The CAU accepts this new configuration from the CU and implements it. Then it initiates the self-check in the new subcomputer. If this test fails, the CAU takes over the complete reconfiguration responsibilities.

The part of the CU in the process is as follows. Upon receipt of the command to initiate a diagnostic, it stores the address of the present instruction, branches to the main memory location of the first statement of the diagnostic, and resets the idle-time counter and diagnostic-time counter. If the diagnostic is successful it stops the diagnostic-time counter. If any error occurs it drops into an idle mode immediately, letting the diagnostic counter go to zero which allows the CAU to take over. At the end of a successful diagnostic the CU either notifies the CAU or resumes operation if the diagnostic was self-initiated rather than CAU-initiated.

The CAU contains 16 registers. Various other units may read or write directly into these registers. The versatility of the G1 character makes it possible for any unit to write or read its assigned registers without calling attention of the CAU. This is possible because the CU-CAU interface is not restricted to 8 bits. The CAU also contains 4 counters which are similar to the registers in their accessing and address characteristics. The CAU has physical control over the byte interface switches. These switches are designed to be addressable and to accept coded state information and act accordingly. The CAU is capable of setting the status register according to information received from discretes from any unit or by CU interrupt.

CONTROL UNIT DIRECTION OF THE SUBCOMPUTER UNITS

The Control Unit (CU) directs the operation of its subcomputer by means of microprogrammed subroutines which respond to instructions and data stored in local memory.

The CU is provided with a local data memory (LDM) capable of holding 64 words of information. Data is transferred in blocks of words to and from the LDM and the Memory Unit (MU) according to special program instructions. Access to the LDM by the CU is considerably faster than to main memory. Use of the LDM makes it possible to avoid accessing main memory each time an instruction is executed.

The CU also has a local program memory (LPM) similar in construction to the LDM but containing instructions rather than data. In normal operation, the CU tracks its progress by storing in fast-access registers the address of the present instruction in both the LPM and the main memory. The instruction itself is accessed from the LPM, which acts as a buffer between main program memory and the CU.

At present the most desirable scheme for loading the LPM is to take advantage of the CU-AU overlap time — that is, rather than letting the CU hang up waiting for the AU to finish a calculation, the time may be used by the CU to update the LPM loading. Any program branch instruction initiates or changes the sequence of the LPM loading.

The CU directs the other four units of the subcomputer as follows. (See facing table for subroutines.)

- Arithmetic Unit — The CU sends some of the operation codes and associated operands to the AU. It accepts the resultants from the AU.

- Input-Output Unit — The CU transfers input and output command words to the I/O. No further participation of the CU in the input-output process is required. The I/O unit establishes a data path between the MU and the appropriate device and performs all functions necessary for completion of the information transfer. The CU may request the status of the transfer from the I/O, which will (at the completion of the current word being transferred) reply by sending the CU the word count (number of words remaining), address in main memory of the next word, and the number of the external device involved.

- Memory Unit — The CU can read or write into main memory one word at a time or in blocks. The CU initializes the MU with a command and address and, after an appropriate delay, accepts or sends the data to the MU. In case of conflict, the I/O unit has priority for use of the memory.

- Configuration Assignment Unit (CAU) — Communication between the CU and CAU differs from the original (MCB) design. The CU can write into or read those registers and counters assigned to it but physically contained in the CAU (mask registers, diagnostic clocks, cross communication registers, etc) without disturbing the flow of operations of the CAU. The CU remains attentive to CAU interrupts and responds to commands from the CAU to initiate diagnostics. The CU can transmit a new configuration to the CAU by a CAU interrupt.

TABLE 3-I. LIST OF CONTROL UNIT SUBROUTINES

Mnemonic	No.	
EA3	1	Calculates effective address (appropriate index register contents plus contents of const. field of instruction = address) when 3 are needed.
EA1	2	Calculates effective address when only one is needed (access a portion of 1).
RAC2	3	Given an address at random (except one in main memory) it accesses the appropriate storage device and reads in the data. Access 2 address.
RAC1	4	Similar to 3 except only the data from 1 address is called in.
RAW	5	Writes data into any address desired.
AU NPT	6	used for communication with the AU.
INT AU	7	
IO NPT	8	used for communication with the I/O.
INT IO	9	
CAU NPT	10	used for communication with the CAU.
INT CAU	11	
INT MUA	12	used for communication with the MU.
INT MU TO MU	13	
INT MU TO CU	14	

FUNCTIONAL DESCRIPTION OF THE CONTROL UNIT

The Control Unit through its versatile, 16-bit double-logic unit controls the operation of the MCB subcomputer.

The Control Unit is a 16-bit double-logic-unit machine. The principal logic unit, which interfaces with local memories, clocks, the MU, and the CAU, has adder (L2) capability. The auxiliary logic unit, which interfaces with the AU and I/O UNITS, has no adders. Program counters, index registers, buffers, and working registers are in the common G1-register bank (Figure 3-4). CAU equipment interfaces with the CU wherever the union is most efficient.

Operation of the CU is best illustrated by its microprogram flow chart (Figure 3-5). The CU executes one instruction per loop of the microprogram. Upon obtaining the instruction from the LPM, it enters the operator field to the micromemory counter to effect the decoding. The CU then proceeds along one of 32 parallel paths which may be grouped into 4 main categories, described below.

If the instruction is an input-output command, the instruction word is immediately transferred to the appropriate I/O Unit. This is the extent of the CU's participation in the I/O process.

If the command is one of 7 that are performed in the AU (floating point or multiply operations), the CU utilizes its subroutine repertoire to calculate the effective address (EA3 or EA1) and access the operands (RAC2 or RAC1). The "Random Address Call" subroutines (RAC) are considerably complex and may use any or all of the CU's capability to access data. The CU then communicates with the AU as follows. If the AU is still processing a previous command, the CU utilizes the overlap time to update the LPM. When the AU becomes attentive, the CU first accepts and stores any answer the AU has available. Then the next operand group is sent to the AU. All AU communication is through the L3 of the auxiliary CU logic unit.

If the command is a local data memory block transfer, it is processed one word at a time. Between words the CU checks for interrupts (notably a CAU interrupt or an AU "answer ready" signal) just as it normally would between instructions. Communication with the MU is by means of subroutines which drive the two L3's of the CU main logic unit. Data passes to and from the LDM through these same L3 cards. Therefore block transfer data bypasses most of the CU logic circuits.

The fourth category is composed of internally processed CU functions (not requiring AU or I/O). Originally this was composed of memory loads, stores, and program control functions (branches). To this have been added the simpler mathematical functions such as "add" (not floating point), "and," "or," etc. The reason for the alteration follows. The generality of the character set results in the CU having much the same logical properties as the AU. This means the CU can perform the simpler functions in less time than it would take to establish an AU communications link. The only drawback is the 16-bit capacity of the CU. All operations must be double-precision, but experience shows that given this character set and the simpler functions, the penalty is small.

In this category liberal use is made of subroutines to calculate address, access data, store answers, and communicate with other units. The CU microprogram example, "direct add," is taken from this category (Figure 3-6).

Before completing the loop and going on to the next instruction, the CU checks for external interrupts.

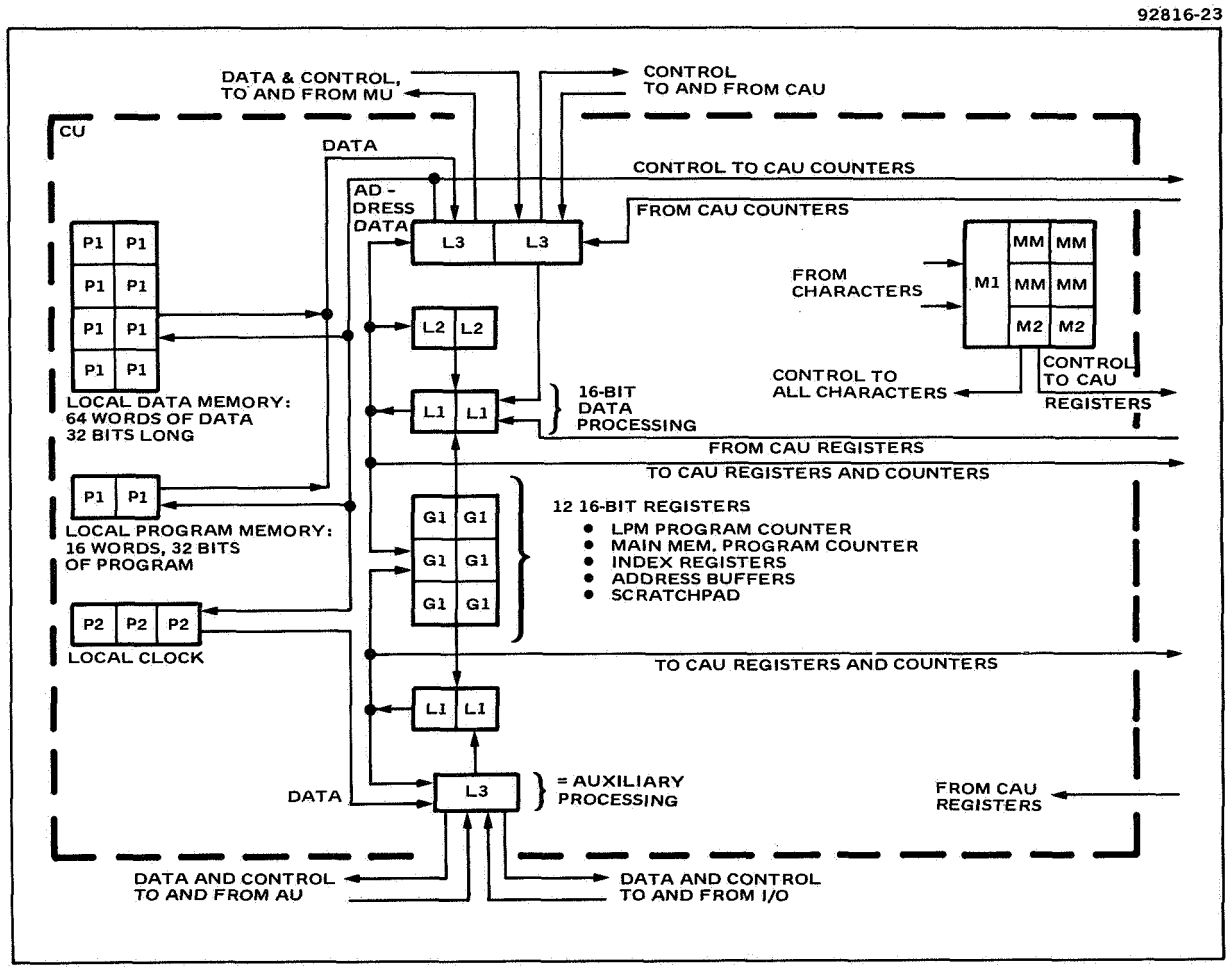


Figure 3-4. MCB Control Unit Block Diagram. Versatility is provided by the 16-bit double-logic unit.

FUNCTIONAL DESCRIPTION OF THE CONTROL UNIT (Continued)

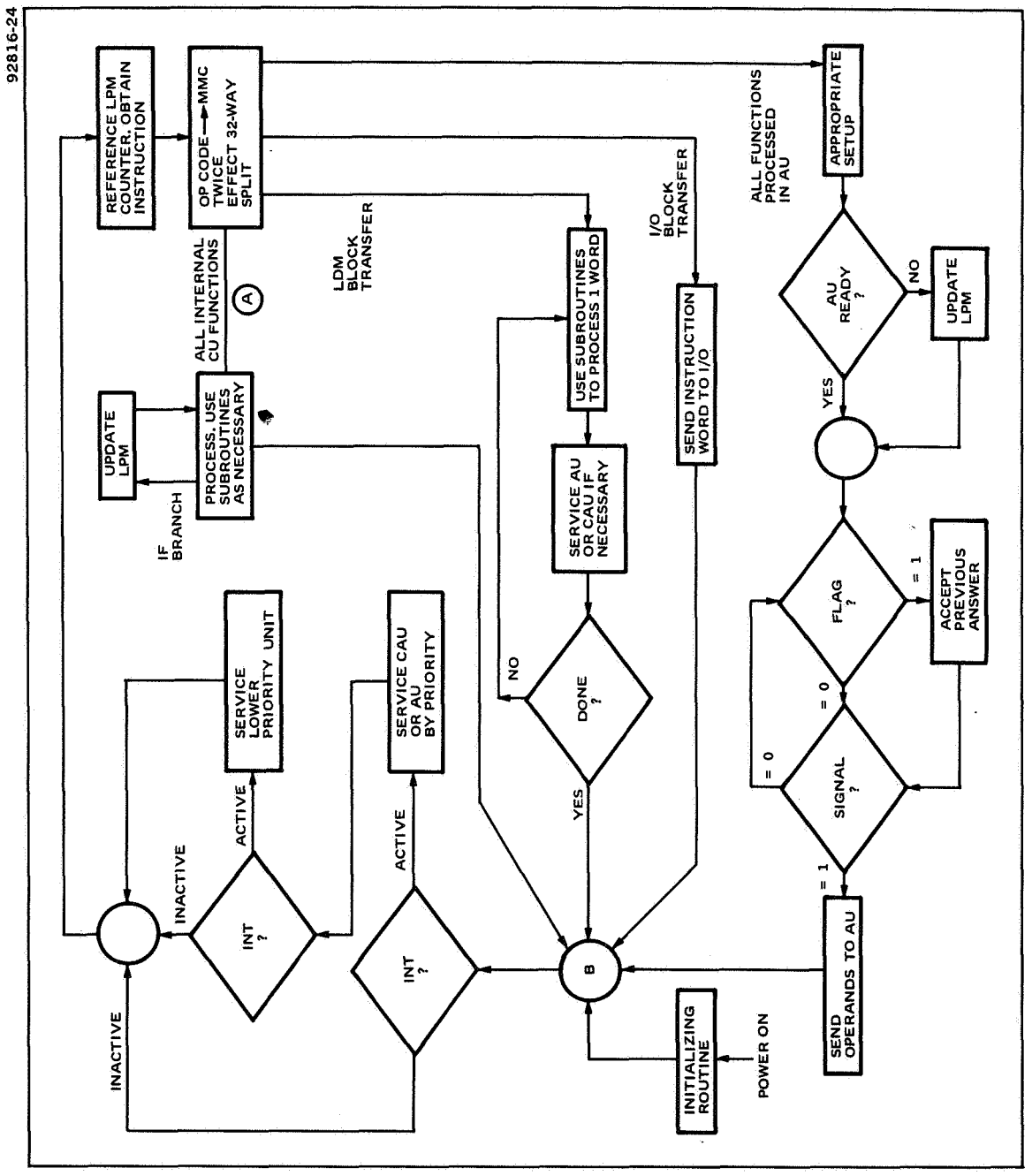


Figure 3-5. Control Unit Microprogram Flow Chart — General. After the instruction is received and decoded, the CU proceeds along one of 32 parallel paths according to the functional category of the instruction.

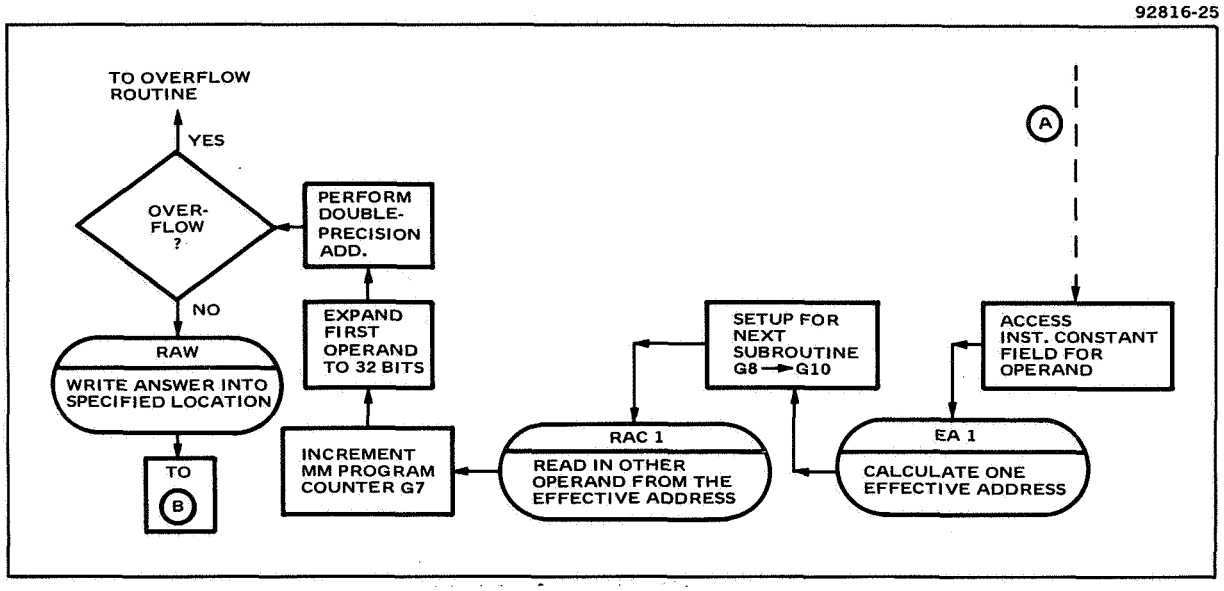


Figure 3-6. Control Unit Microprogram Flow Chart — Detailed Example. This "direct add" microprogram is one of the internally processed CU functions.

FUNCTIONAL DESCRIPTION OF THE ARITHMETIC UNIT

The Arithmetic Unit is a 32-bit, single-logic-unit machine which is responsible for seven arithmetic instructions.

The Arithmetic Unit consists of four L1 and four L2 cards providing 32-bit logic and adder capability, one L3 for CU communications, and G1 cards for processing and operand and answer buffering. The AU performs the following instructions: multiply, most and least significant halves, floating point multiply, divide, add, subtract; and add-not-normalized.

The Arithmetic Unit is a 32-bit single-logic-unit machine. For more complicated functions, this arrangement, rather than that of the CU, is optimal. Four L1 and four L2 cards provide 32-bit logic and adder capability. One L3 is sufficient for CU communications. One 8-bit register serves as a command buffer and three 32-bit registers as operand and answer buffers; the remainder are working registers. (See Figure 3-7).

As illustrated by the microprogram flow chart (Figure 3-8), the AU utilizes command decoding and parallel branches similar to those of the CU. The AU's responsibility has been reduced, consisting now of the following seven instructions: multiply, most and least significant halves; floating point multiply, divide, add, subtract; and add-not-normalized. Each loop processes one instruction only.

Outside of setting "answer ready" and "empty" flags to the CU, the AU is passive, responding to CU interrupts. Except for start-up, the AU will not process until it has received two CU interrupts. One is for the answer and the other for new operands; order does not matter. Programming and timing make it impossible for the CU to attempt the same subroutine twice in succession.

92816-26

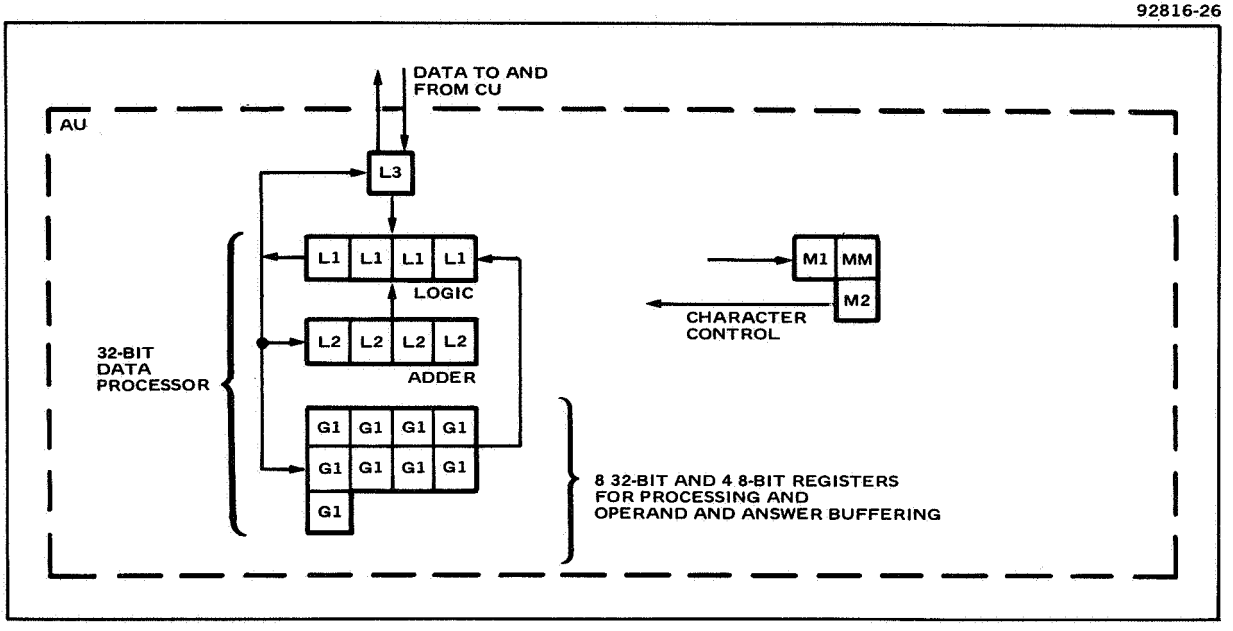


Figure 3-7. MCB Arithmetic Unit Block Diagram. With its 32-bit single-logic unit, the machine is better than the CU for more complicated functions.

92816-27

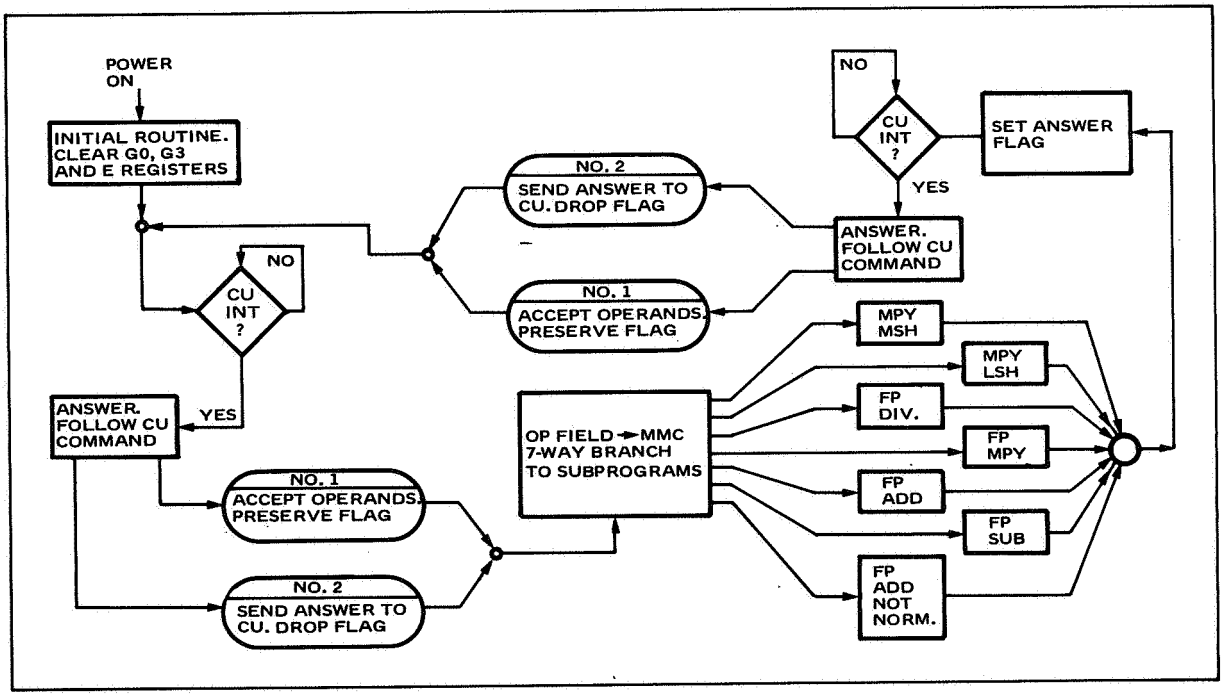


Figure 3-8. Arithmetic Unit Microprogram Flow Chart. The Arithmetic Unit is responsible for seven arithmetic functions.

FUNCTIONAL DESCRIPTION OF THE MEMORY UNIT

Each Memory Unit contains one main memory module capable of storing 4,096 words, and the few characters needed to transport data and operate the memory module.

The memory unit is the simplest unit from a character standpoint. It carries only a minimum of characters to transport data and to operate the memory module (Figure 3-9). It is desirable to have main memory address and data registers that can link to LI and operate under micromemory control.

The flow chart (Figure 3-10) indicates that the normal mode of the MU is to remain attentive for external interrupts. Upon receipt of interrupts, the MU will respond first to the I/O. Otherwise it responds to the CU. In an operation carefully timed to allow maximum overlap, the MU accepts the address, then branches according to the unit's command. Contained in the subroutines are steps to operate the memory module and transfer data to and from the units.

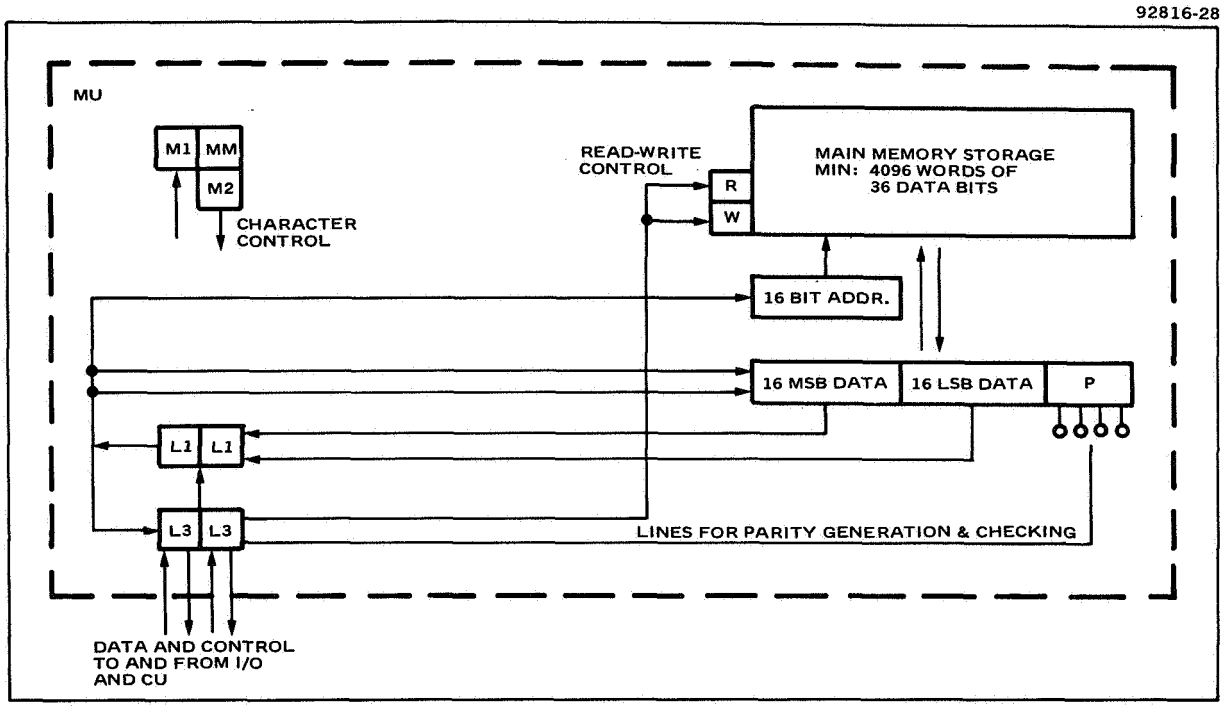


Figure 3-9. MCB Memory Unit Block Diagram. A minimum of characters are required.

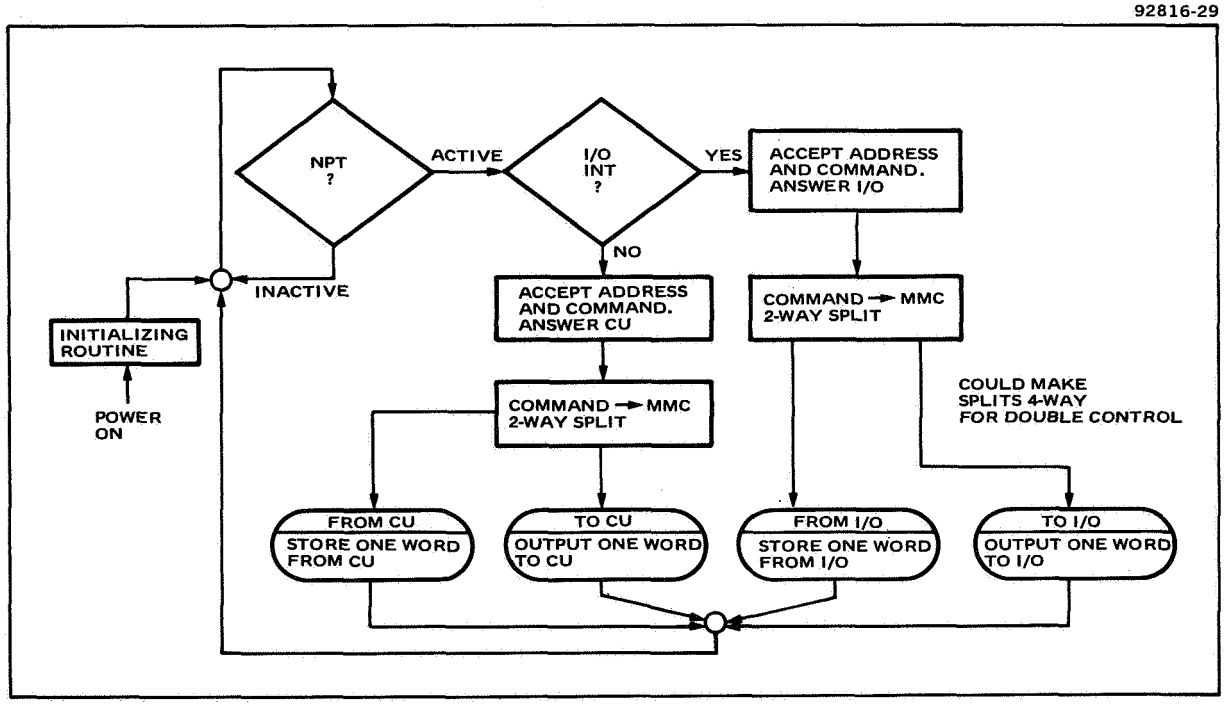


Figure 3-10. Memory Unit Flow Chart. The normal mode is to wait for external interrupts.

FUNCTIONAL DESCRIPTION OF THE INPUT/OUTPUT UNIT

The Input/Output unit, which interfaces with the memory unit and all I/O devices, requires the most intricately sequenced microprogram.

The Input/Output Unit can control the Memory Unit and all input/output devices. The characters provide a 32-bit data interface with external devices and can carry out an entire block transfer.

Four L3 cards are used almost to their full capacity to provide an interface with four input devices, two output devices, the CU, and the appropriate MU (Figure 3-11). Two of the inputs, referred to as "No. 0" and "No. 1," are MCB-interrupting inputs which, to the best of our knowledge, do not exist in the current version of the MCB. Their presence requires two G1 cards to store the addresses of their inputs in main memory. The remaining four I/O devices are those covered by the I/O block transfer word, stored in two of the 32-bit registers. The remaining registers are scratchpad.

The microprogram of the I/O is the most intricately sequenced of any unit (Figure 3-12). The idle mode is represented by the waiting-for-interrupt loop in the middle left edge of the figure. Upon receipt of interrupts, input unit No. 0 has priority. The upper line of the flow chart is the branch servicing unit No. 0 (storing its input in main memory). Input unit No. 1 has next priority. Its servicing branch is nearly identical to that of No. 0.

The CU has lowest priority. A CU interrupt means either a replacement of one of the block transfer commands or a request for the current status of one of the block transfer commands. Consequently a CU interrupt initiates a four-way decision according to a few key bits. 10 and 11 are shown as indicating requests for the priority word (G0) or the normal word (G1) to be sent to the CU. A subroutine accomplishes this. 00 and 01 indicate new transfer commands. If the old command is not done the new one supersedes it, allowing the CU some I/O control.

The command in G0 has priority, and is always carried to completion before the command in G1 is considered. When the block transfer is done, the command register is made zero, indicating completion.

The block transfers occur within the heavy outline on the main chart and the 00 block transfer path is detailed in Figure 3-13. The first basic decision is whether the command is input or output. Then a data path is established between the MU and the device using appropriate subroutines, signals, and the common L3 bank. The transfer ends when the word count reaches zero. Each loop of Figure 3-13 transfers one word. Interrupts are accepted between each word transferred.

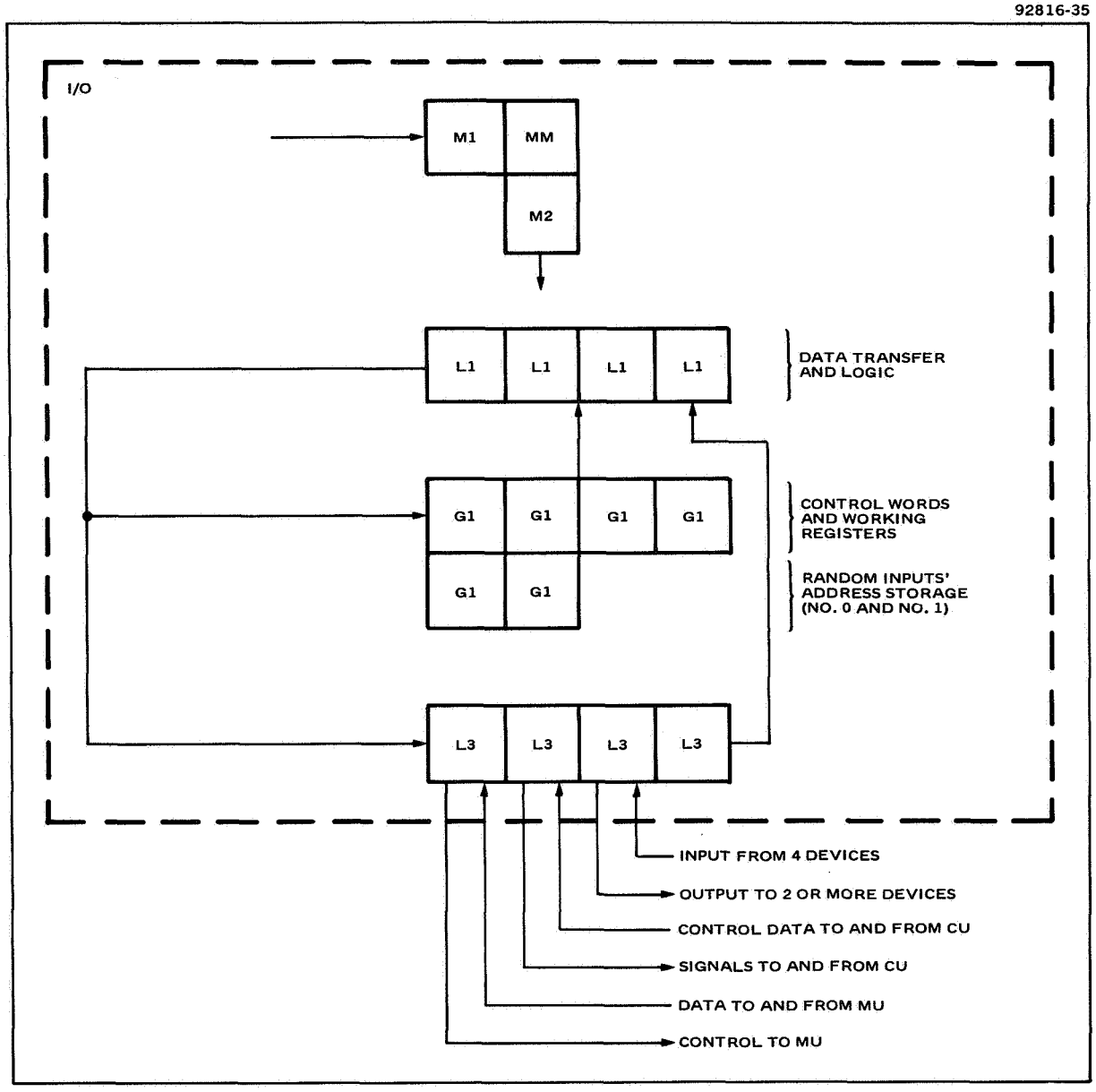


Figure 3-11. MCB I/O Block Diagram. The characters provide a 32-bit data interface with external devices.

FUNCTIONAL DESCRIPTION OF THE INPUT/OUTPUT UNIT (Continued)

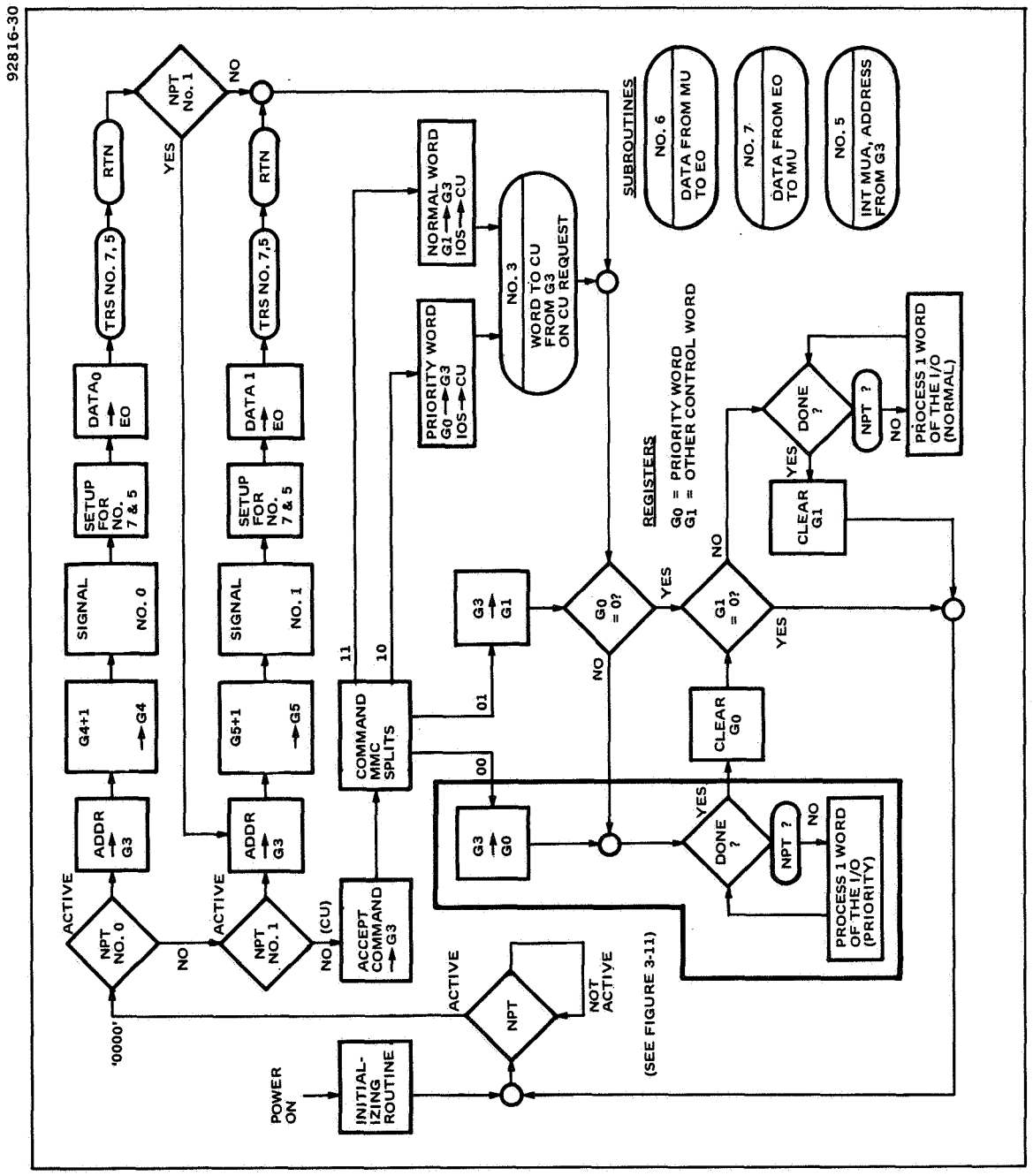


Figure 3-12. I/O Microprogram Flow Chart — General. This microprogram is the most intricately sequenced of any unit.

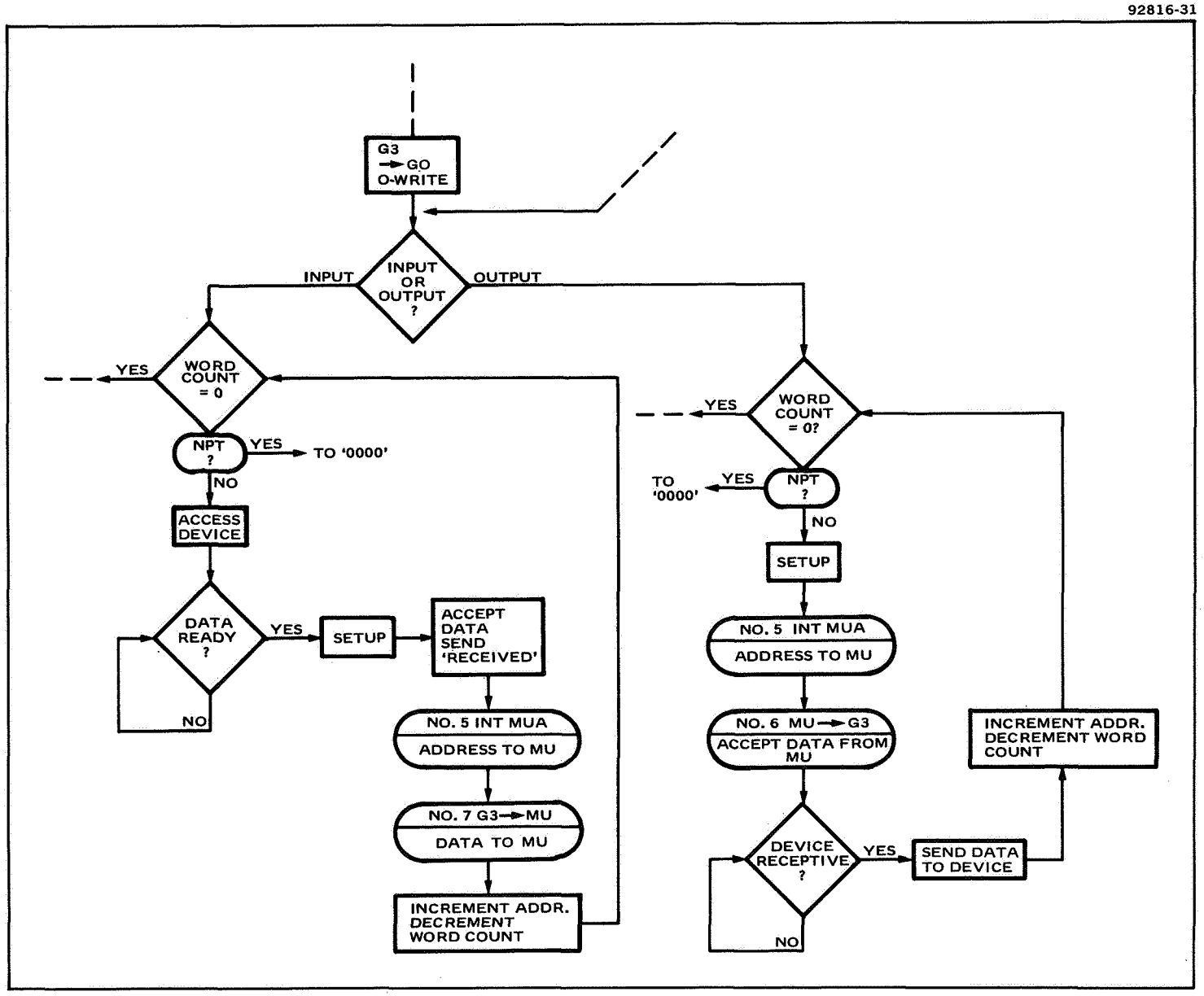


Figure 3-13. I/O Block Transfer Microprogram Flow Chart — Detailed Example. This flow corresponds to the area enclosed in heavy lines in the facing figure.

FUNCTIONAL DESCRIPTION OF THE CONFIGURATION ASSIGNMENT UNIT

The Configuration Assignment Unit (CAU) monitors the idle-time counter and the diagnostic-time counter, ready to take over the reconfiguration duties if the counters go to zero. Also, the CAU, rather than the CU, compares mask and status registers.

Not all of the characters physically located in the CAU (Figure 3-14) are under control of the CAU micromemory.

Although physically located in the CAU, the cross-communication registers, four of the status registers, and three of the counters are under CU control. The remainder of the CAU equipment is either uniquely under CAU control, or controlled jointly with one or more CU's.

Although designed to function in a manner identical to the original MCB, the CAU shows the most deviation of any unit from the original implementation. The major change is that it is now the CAU rather than the CU which compares status and mask registers. The remainder of the alterations are minor.

The CAU utilizes two L3 characters to operate counters and switches, accept discrete inputs, and communicate with the CU.

The microprogram of the CAU is relatively simple, concerned mainly with checking for errors (Figure 3-15). The upper loop of Figures 3-15 is the normal mode, representing no errors detected. If an error is detected, its nature determines which of two error-correcting branches the CAU will enter.

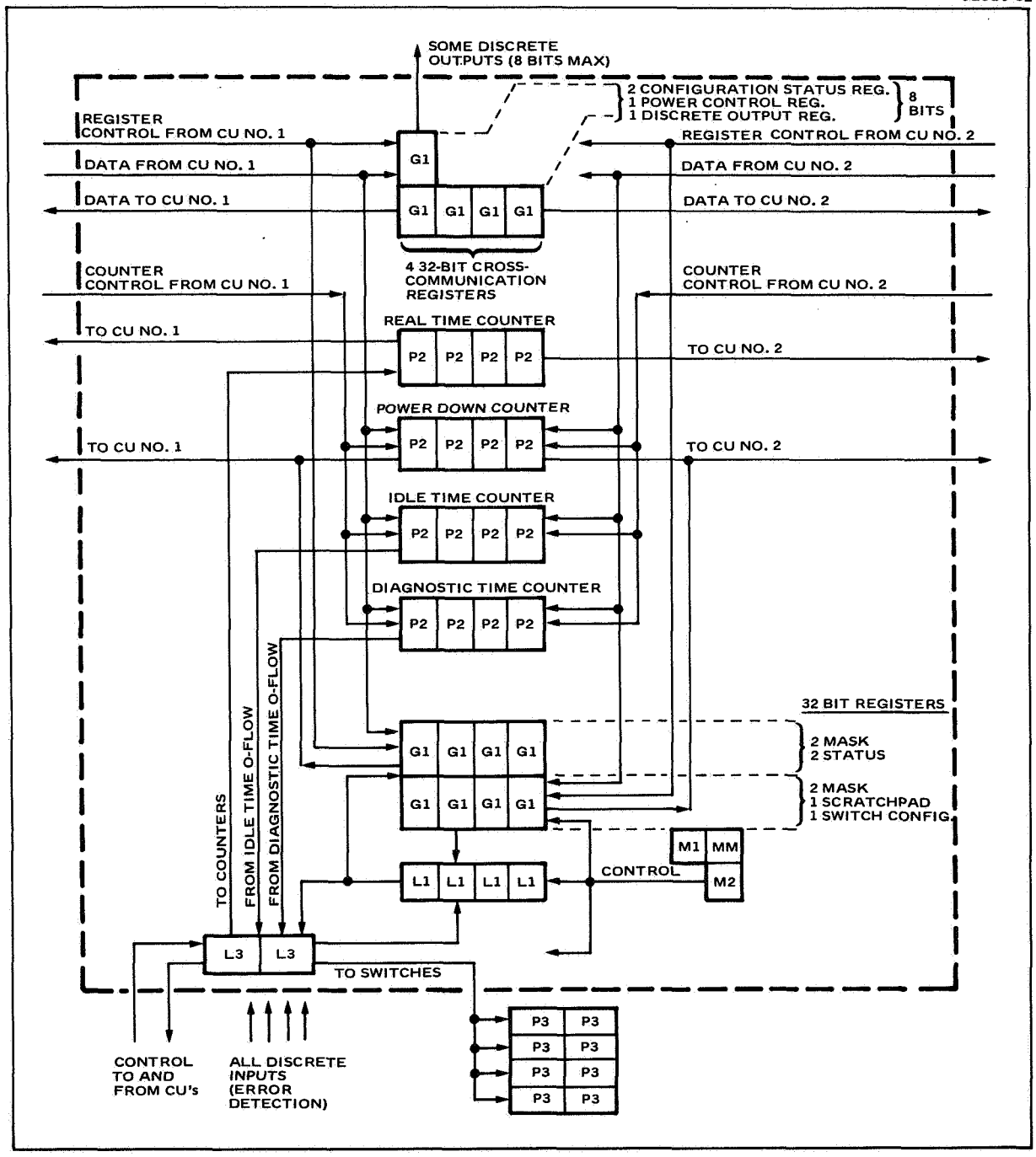


Figure 3-14. MCB Configuration Assignment Unit Block Diagram. The CAU Takes over reconfiguration duties if the idle-time counter and diagnostic-time counter go to zero.

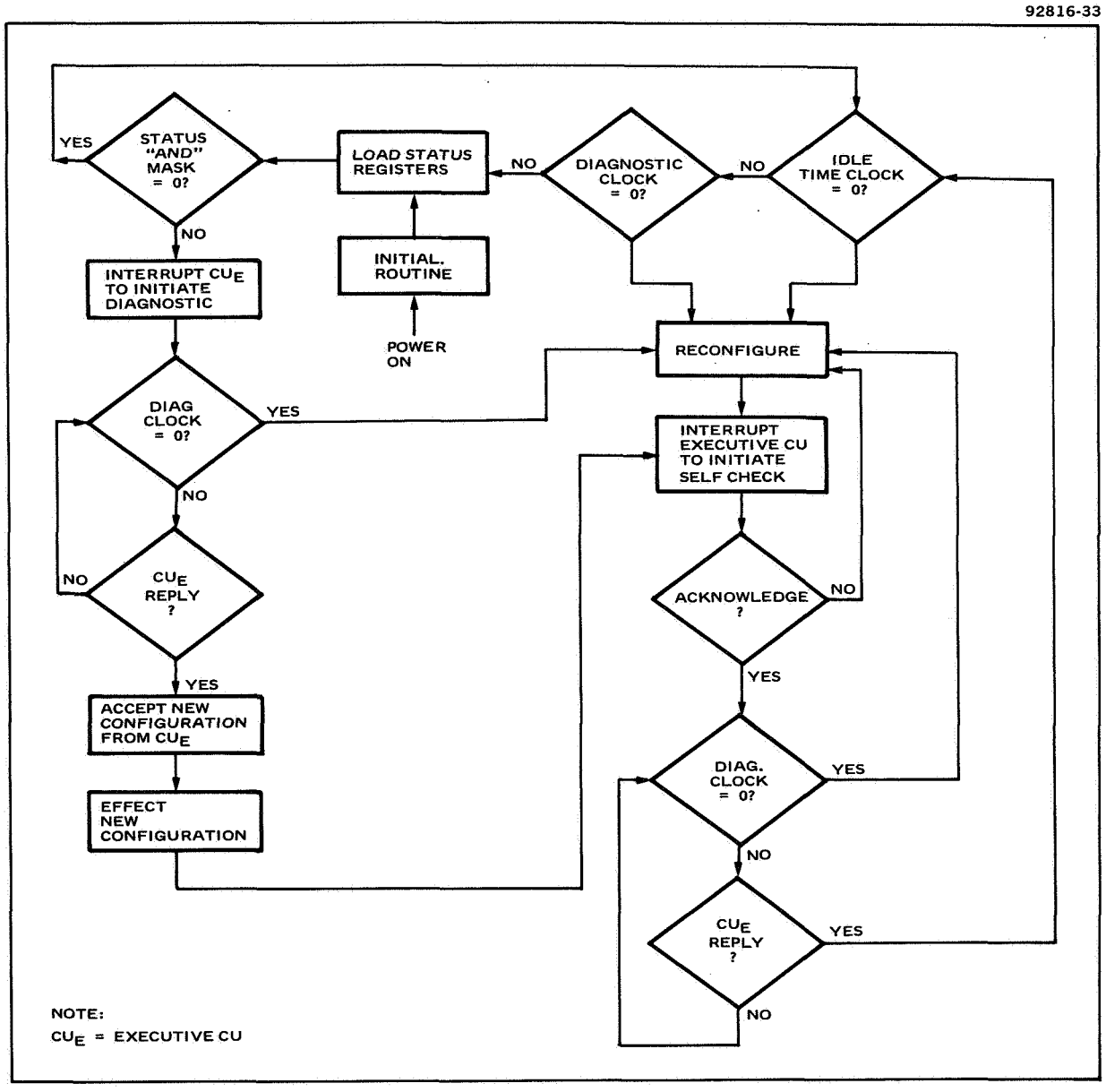


Figure 3-15. Configuration Assignment Unit Microprogram Flow Chart. This is a relatively simple microprogram mainly concerned with checking for errors.

COMBINED CONTROL AND ARITHMETIC UNITS IN AN ALTERNATIVE CONFIGURATION

The combined Control-Arithmetic Unit is a 32-bit double-logic-unit machine possessing fewer characters than the units do separately, less inter-unit wiring, and faster capability for complicated arithmetic functions.

Study of the MCB has included a system in which the remainder of the AU functions are absorbed by the CU. This means one unit, illustrated in Figure 3-16, replaces a CU and an AU. The resulting system has two I/O, two MU, one CAU, two of the new unit, and 6 switches.

The combined CU and AU (CUAU) is a 32-bit double-logic unit machine. The more complicated AU functions require a full-word-length logic organization for maximum efficiency. More registers are also required for scratch-pad. The lack of an AU interface allows a reduction in the number of L3 characters. The structure of the new unit more closely resembles that of the CU, taking on only the word length and the G-register capacity of the AU.

An estimate of the speeds of the CUAU predicts that more complicated functions such as multiply and floating points can be done 2 to 3 microseconds faster. Time saved on the simpler functions ranges from 0 to 0.5 microsecond.

The more significant change due to this structure is in the hardware count. One CUAU contains 45 characters. This replaces a CU with 35 characters and an AU with 21. Considering that two P3 (switch) characters are saved, this reduces the total MCB character count from 206 to 182. It also reduces the MCB inter-unit wiring up to 25 percent of its previous value.

Reliability is an open question. Although it is true that the hardware count goes down, this is offset by the fact that redundancy is lost (the CUAU is larger than either the CU or the AU alone). Also of importance is the elimination of the data transfer between the CU and the AU.

The CUAU microprogram is similar to that of the CU (see page 3-12). The major difference is in the program branches previously referred to as "functions processed in the AU." Parts of these are now replaced with branches similar to the seven of the AU. Also, the CUAU need only be attentive to one interrupt per loop. All subroutines and program steps relevant to AU communication are dropped. The increased capability is used to maximum efficiency elsewhere in the program.

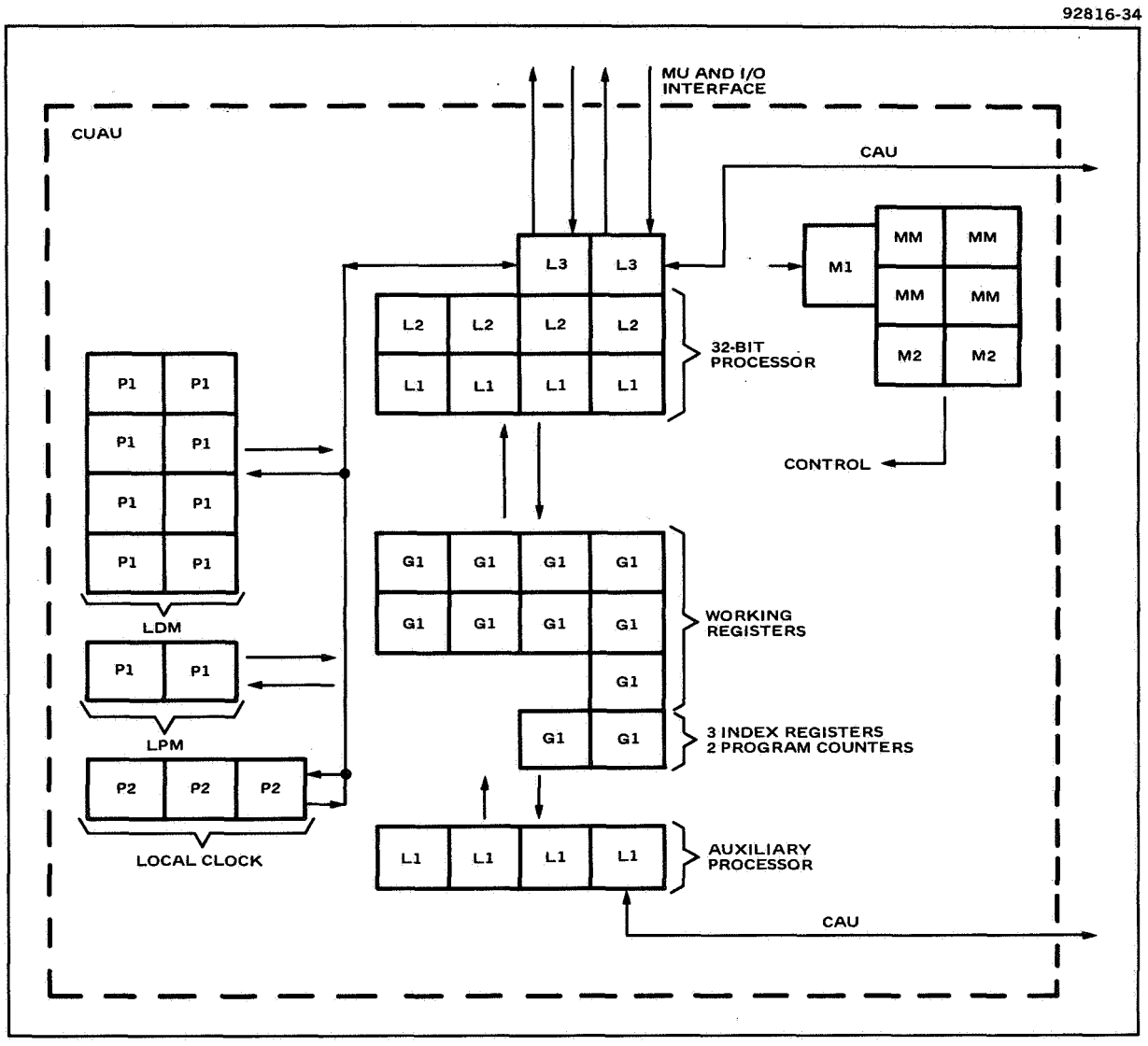


Figure 3-16. CUAU Block Diagram. The combined unit is a 32-bit, double-logic-unit machine which is fast for complicated arithmetic functions.

EXECUTION RATES FOR THE HUGHES AND EXISTING VERSIONS OF THE MCB

Design of the MCB using functional characters has resulted in no speed compromise. In fact, the Hughes version is faster in execution of most instructions.

The facing table contains the execution times for the MCB and the existing version of the MCB. These times are based on an assumed micromemory cycle rate of 5 MHz for the Hughes-designed machine. Logic runs at 10 MHz. Times are for the original 5-unit machine, not the system with combined CU and AU.

Note that execution times are faster with the character implemented machine for all instructions except "Branch Direct" and "Decrement and Branch."

TABLE 3-II. MCB EXECUTION TIMES

	Hughes Version (μsec)	Existing MCB (μsec)
All Shifts	5.8	14.5
Add Direct (Fixed Point)	4.6	11.0
Add (Fixed Point)	6.8	11.0
Subtract (Fixed Point)	6.8	11.0
Multiply, Most Sig. Half (Fixed Point)	20.8	29.0
Multiply, Least Sig. Half (Fixed Point)	20.8	29.0
Inclusive OR	6.2	9.5
Exclusive OR	6.4	10.0
Logical AND	6.2	9.5
Floating Point ADD	5.4 + Δ	10.5 + Δ
Floating Point Add Not Normalized	5.2	11.0
Floating Point Subtract	5.4 + Δ	10.5 + Δ
Floating Point Multiply	21.8 + Δ	26.5 + Δ
Floating Point Divide	21.0 + Δ	51.0 + Δ
Store Direct	4.8 + W	5.5 + W
Load Direct	5.3 + W	5.5 + W
Store	7.0 + W	8.5 + W
Load	7.7 + W	8.5 + W
Block Transfer Setup	1.2	1.5
Block Transfer Per Word	3.0 + W	5.0 + W
Input Output Setup	2.0	4.5
Input Output Per Word	3.8 + W	7.0 + W
Branch Direct	3.4 + W	2.0 + W
Conditional Branch	3.8 + W	4.0 + W
Decrement and Branch	3.6 + W	3.0 + W
Branch and Link	4.6 + W	5.0 + W

W = memory wait time
Δ = equalization + normalization time

For the Hughes version:

equalization = 1.9 μsec (AVG)
normalization = 3.3 μsec

EVALUATION OF THE FUNCTIONAL CHARACTER DESIGN OF THE MCB

In terms of LSI circuit types, total pins, and gate-to-pin ratio, the Functional Character implementation is superior to a conventional logic implementation of the MCB.

Table 3-III shows the comparison data of the functional character implementation versus the existing MCB implementation. As may be seen, in all aspects, except gates committed, the functional character implementation results in a significant improvement over the existing MCB design. The number of gates committed is 35 percent higher for the functional character approach. In the LSI area, the tradeoffs will no doubt recognize the functional character approach as significantly superior. An increase of 35 percent in the number of gates committed is a small price to pay for the reduction in the number of chip types and pins.

For reliability purposes, a small number of pins in the system is far more important than a small number of gates, all other factors being equal. As seen in Table 3-III, the number of pins required for the MCB implementation is 2.6 times the number required for the functional character implementation.

As the column heading shows, the comparison in Table 3-III is made between two LSI implementations: one representing the functional character technique, the other representing the conventional approach where every MCB card containing X number of IC's has been converted to an equivalent IC with the number of card terminals becoming the equivalent LSI package pins.

Design with functional characters saves time. During a six month period, the entire MCB was designed, microprogrammed, and remicroprogrammed several times. This illustrates the ease and speed of the design process. The improvements gained through microprogramming are demonstrated in Tables 3-IV and 3-V.

Table 3-IV shows the improvements in terms of the number of characters and character types required for the two microprogram passes. The characters remained unchanged; In this comparison, the configuration of the MCB was identical with the presently implemented IC version.

Further improvements were gained, as shown in Table 3-V, by restructuring the MCB with the appropriate remicroprogramming. The AU and CU were combined into one unit, eliminating some logic and the switch between them. This reimplementation was feasible with the functional character set due to the more general nature of the characters as contrasted with the custom implementation of the existing MCB.

TABLE 3-III. COMPARISON OF FUNCTIONAL CHARACTER IMPLEMENTATION AND EXISTING MCB IMPLEMENTATION

Item \ Implementation	Functional Character Implementation (Units)	MCB Assuming Each Card Is An LSI Chip	Percent Improvement Over MCB Implementation	High/Low Ratio
Types	10	23	+56	2.30
Cards (LSI Chips)	206	554	+63	2.70
Pins Committed	18,200	47,600	+62	2.62
Gates Committed	47,200	35,000	-35	1.35
Gates/Pin	2.6	0.75	+250	3.47

TABLE 3-IV. EFFECTS OF MICRO-PROGRAM IMPROVEMENT ON THE FUNCTIONAL CHARACTER IMPLEMENTATION OF THE MCB

Unit	No. of Characters Used		No. of Character Types Used	
	MICROPROGRAM PASS		MICROPROGRAM PASS	
	First	Subsequent	First	Subsequent
MU	11	7	6	5
CAU	39	38	7	7
CU	38	35	9	9
AU	25	21	7	7
Switches	8	8	1	1
I/O	17	17	6	6
Computer Total System	229	206	10	10

TABLE 3-V. EFFECTS OF COMBINING THE AU AND CU OF THE MCB

Parameter	Functional Character Implementation of the Existing Configuration	Same Except AU and CU Were Combined
No. of Characters	229	182
No. of Character Types	10	10
Fixed Point Direct Add	9.9	4.2 us
Fixed Point Add	11.6	6.4
Fixed Point Subtract	11.6	6.4
Inclusive or	11.5	6.2
Exclusive or	11.5	6.4
Logical and	11.5	6.2

BLANK PAGE FOLLOWS 3-31/3-32

SECTION 4
CHARACTER SUB-PARTITIONING

Section 4 – Character Sub-Partitioning

ALTERNATIVE SCHEMES FOR SUB-PARTITIONING

Alternative schemes for sub-partitioning the characters in accordance with NASA guidelines has yielded partitions of 150 gates or less but with degraded gate-to-pin ratios.

Some functional characters require about 350 gates per function. The natural tendency would be to implement one character per chip. However, this is not an acceptable solution for TTL circuits in view of the state of the art and the following NASA-desired design guidelines.

Gates per chip	– About 100, no more than 150
Circuit yield	– 100% without yield enhancement
Conductor spacing	– 0.1 mil minimum
Conductor width	– Current density not to exceed 10^5 amps/cm^2
Metalization layers	– No more than 2
Circuit type	– Bipolar TTL

In view of these guidelines, the functional characters were sub-partitioned as shown in the table.

The intent of the table is not to select the optimal subpartition, but to enumerate some logical choices. The optimal choice will depend on assigned weightings for gates and pins per chip, as well as the other design constraints mentioned earlier. The table thus shows each character and the characters' composition, using one or more custom or commercially available LSI/MSI chips. More than one subpartitioned chip is required to implement the functional character. The number of chips and chip types required is given in the second column as a descriptor and also in the sixth and seventh columns under "composite." The columns under the "composite" heading state the total number of items required to implement one functional character. The columns under the first and second chip heading contain similar information on a per chip basis.

A comparison the subpartitions with the character partitions shows the following changes:

1. The number of chip types is at least 20% greater than the number of characters; thus, a small penalty is paid in terms of part number problems.

2. The number of gates per chip dropped (approximately by a factor of 0.5) and the number of pins remained about equal, resulting in an increased number of pins in the system by a factor of about 2.

3. The total number of gates per function increased an insignificant amount.

TABLE 4-I. ALTERNATIVE SCHEMES FOR SUB-PARTITIONING

Character		Composite					1st Chip				2nd Chip			
Name	Composition	Gates	Pins	G/P Ratio	Chips/ Character	Chip Types	Gates/ Chip	Pins/ Chip	Ratio Gate/Pin	No. Used	Gates	Pins	Gate/Pin Ratio	No. Used
G1-Register	2 custom chips, single type	224	62	3.6	2	1	112	52	2.2	2				
L1-Logic	2 custom chips, single type	274	145	1.9	2	1	137	138	1.0	2				
L2-Adder	2 Identical custom chips	258	77	3.4	2	1	129	60	2.2	2				
	1 custom and 1 commercial chip	228	77	3.0	2	2	117	88	1.3	1	111*	43*	2.6*	1*
L3-Input/Output	4 identical chips	456	150	3.1	4	1	114	72	1.6	4				
	2 identical chips with optional parity	410	149	2.8	3	2	150	87	1.7	2	110	68	1.8	1
	Alternate Scheme 2 chips + optional parity chip	398	149	2.8	3	2	150	95	1.6	2	98	66	1.5	1
	Optimal 3-chip configuration	377	149	2.5	3	2	129	85	1.5	2	119	86	1.4	1
M1-Micromemory Sequencer	3 chips-2 types	358	91	3.9	3	2	150	92	1.6	1	104	85	1.2	2
	3 chips-2 types 150 gates if I.C.	348	91	3.8	2	2	142	73	1.9	1	206	73	2.8	1
P2-Counter	1 custom chip	148	81	1.8	1	1	147	81	1.8	1				
	1 custom and 2 commercial chips	163	81	2.0	3	2	40*	*	*	2*	83	82	0.9	1
P3-Switch	2 identical chips	210	118	1.8	2	1	105	75	1.4	2				
M2-Micro Instruction Register	3 chips-2 types	323	131	2.5	3	2	100	51	2.0	2	123	89	1.4	1

*Commercially available chip

RELIABILITY CONSIDERATIONS IN CHARACTER PARTITIONING

The achievement of maximum reliability depends on minimizing operating temperature and total number of pins in the system. Subpartitioning the characters will tend to lower reliability because of increased pin count.

Reliability considerations require a minimum number of bonds (pins) and a lowest junction temperature practicable. Temperature is a very important consideration since the failure rate of the device increases about 1.8 times per 25°C temperature rise. Although several other factors affect reliability, they are either less influential on the operational failure rate, or on a relative basis do not affect the tradeoff. For example, the quality of the package's hermetic seal may be an important factor in development and acceptance testing. But once a good seal has been established, it will remain good. Furthermore, the difficulty of making a good seal is proportionate to the lengths of seal interface. The latter in turn is a function of the number of pins per package, and the dependence of reliability on number of pins again appears.

Within specific cooling capacity, circuit technology, and packaging technology, two factors affect the device's temperature:

a. The number of gates per system
b. The number of IC package pins per system.

The number of pins affects device temperature because LSI circuits are generally built with tailored lower power internal gates for driving low capacitance and limited fanout within the chip's boundaries and higher power gates at the chip output in order to overcome the input output capacitance and chip fanout.

For example, in the natural and subpartitioned functional characters the number of gates per system remains approximately constant. However, the number of pins doubled for the subpartitioned case. In TTL circuits the power dissipation of the subpartitioned implementation is expected to increase. Specifically, the dissipation is increased by about 15% since the total number of functional character pins increases by more than a factor of two in the subpartitional case. The increased number of pins affects total power since each pin requires at least one external gate, and each external gate dissipates at least 2p units of power. Typically, 1/3 of the pins are used for output; the others are used for inputs, power, and ground. Thus, the total power is given by the product of the power per gate and the total number of gates plus one-third the number of pins, $P = p\,(M + \frac{N}{3})$. This calculation is made for the three cases of interest in the facing table.

Even though the number of gates is 35% greater for the functional character implementation, the power dissipation is only about 5% greater than that of the conventionally-implemented MCB were it implemented with LSI's representing present MCB cards. This 5% difference will disappear in practice. The actual power difference relative to the present IC implementation would be in favor of the functional implementation.

In addition to the number of pins causing increased power dissipation, which may be equated with increased failure rates, there are other reliability and cost penalties associated with an increased number of pins. These all result from bonding. Each pin requires two internal bonds (one to the metalization, the other to the pin). Each pin must in turn be fastened to some external holder (card, connector, wire, etc.).

Every one of these junctions is a potential failure and a fabrication cost factor. Thus, the number of pins as a contributor to increased system failure rate manifests itself in several ways. Every effort must be made to keep the pin count low.

Relative Power Dissipation for the MCB in Three Implementations: Functional Character, Subpartitioned Functional Character, and Conventional Logic: Subpartitioning the Characters Increases Total Power by About 8%.

Basic Formula: $P = p\,(M + \frac{N}{3})$

P = Total Power (watts)
p = power per internal gate
M = total number of gates
N = total number of pins

Case	M	N	P	Relative Power
Functional Character	47,200	18,200	53,300p	1.05
Subpartitioned Characters	47,200	31,500	57,700p	1.13
Conventional Logic	35,000	47,600	50,900p	1.00

SECTION 5
CELLULAR ARRAY MECHANIZATION

Section 5 – Cellular Array Mechanization

CELLULAR LOGIC AS A POSSIBLE MEANS OF CHARACTER IMPLEMENTATION

Although the character set could be implemented with cellular logic, the total gate usage would be prohibitive.

Size, power, and reliability constraints demand that the modular computer be implemented with LSI circuits, but the question of how to achieve an LSI implementation remains. To date, several approaches to logic partitioning for LSI have been reported, ranging from the conventional approach, where partitioning is done after the logical equations have been written, to the "cellular" type approach, where a group of logical gates are structured to be programmed on the cell to form specific functions.

The conventional approach includes both manual and automatic partitioning. This approach appears undersirable for the modular computer implementation because the design process tends to be lengthened and the number of LSI chip types tends to increase, particularly as applications are broadened outside of the computer proper. A small number of LSI chip types is an important factor towards achieving the very tight quality and process controls required for the realization of very low component failure rates. The latter is a must for long time mission reliability.

Cellular logic has been studied for its potential application to the functional characters.

In the cellular approach, the cell design is such that all combinations of n variables must be implementable in order for the cell to be of universal use. Proofs have been developed showing that such a cell can indeed form all functions of n variables. The cell, although a universal device, still requires the process of writing logic and determining which paths in the cell structure should be connected or cut (physically or logically) in order for the universal cell to assume the unique logic posture specified by the logic designer.

The critical deficiency of cellular logic in implementing the characters is that the process of achieving a prespecified function requires disabling of a large percentage of the internal gates. The characters have predetermined logic interconnects but do not require restructuring of interconnections in order to achieve the logical design objective. The design process with functional characters is analogous to programming using a compiler. The characters are analogous to compiler statements. The designer specifies inputs, outputs, and control for each character's micro-operation. Micro-programming is used as the control structure. Three of the 10 characters comprise the micro-program store. Perhaps designing with pre-specified large functions without the utilization of Boolean equations marks the greatest departure and contribution of the functional characters.

The following topics describe the survey of cellular logic which was accomplished. The major areas of investigation are shown in the table.

AREAS OF INVESTIGATION OF CELLULAR LOGIC. THESE ARE DISCUSSED IN THE FOLLOWING TOPICS

- Cellular Array Versus Master Slice
- Cellular Cascades
- Cutpoint Cellular Arrays
- Cobweb Cellular Arrays
- Cellular Mechanizations

THE APPLICATION OF CELLULAR LOGIC TO FUNCTIONAL CHARACTER MECHANIZATION

Cellular logic is a functional character mechanization being considered to reduce the interconnection complexities of large numbers of logic gates. The cellular casade is first described as the forerunner of cellular arrays.

The interconnection of from 200 to 500 logic gates on a single semiconductor wafer in general requires a fairly complex wiring interconnection pattern. In order to reduce the interconnection complexity, some of the connections may be made among the gates to create a basic cell which absorbs some of the next level wiring connections.

One such cell was proposed by Minnick. In its initial form, it was called a cutpoint cell. Its design is rooted in the theoretical decomposition of switching functions. Its varieties and off-shoots are described along with other cellular array types in reference 1. An important variety of the cutpoint cell is the cobweb array cell. It was invented to overcome some interconnection deficiencies associated with the cutpoint cellular array.

Cellular Array Vs. Master Slice – This section assesses the applicability of cutpoint and cobweb cellular arrays to the mechanization of functional characters. A significant part of a typical functional character is mechanized using cellular logic. This mechanization is compared to a master slice mechanization. The master slice mechanization is one which is finding acceptance in the semiconductor industry today.

Cellular Cascades – As a preliminary to understanding cutpoint and cobweb cellular arrays, their forerunner, the cellular cascade is first described. The cellular cascades studied by Maitra (Ref. 2), Sklansky (Ref. 3), and Levy, Winder, and Mott (Ref. 4), are unilateral, one-dimensional, chains of two-input/one-output binary cells. The three approaches accomplish essentially the same purpose by alternate synthesis methods. Maitra's method is one of mapping. Sklansky's method uses matrices. The method of Levy, Winder, and Mott, is algebraic. In light of more recent developments in cellular logic, this type of logic realization might better be classified as a single rail cellular cascade. Its diagram is shown in the figure. Each of the binary functions F_1, F_2, F_n, may be independently set to a function of its two input variables. Maitra, in his original description of the single rail cellular cascade, showed that the set of functions $Q(X_0, X_1, X_2, X_3, \ldots, X_n)$ is smaller than the set of n-variable Boolean functions.

Single rail cellular cascades with redundant inputs have been studied by Minnick (Ref. 7) and Stone, and Korenjak (Ref. 5). These redundant single rail cellular cascades result in a larger class of realizable functions, but for all $m \leq 3$, the entire set of m-variable functions is still not producible.

Single rail cobweb and cutpoint cellular arrays are described in the following topics.

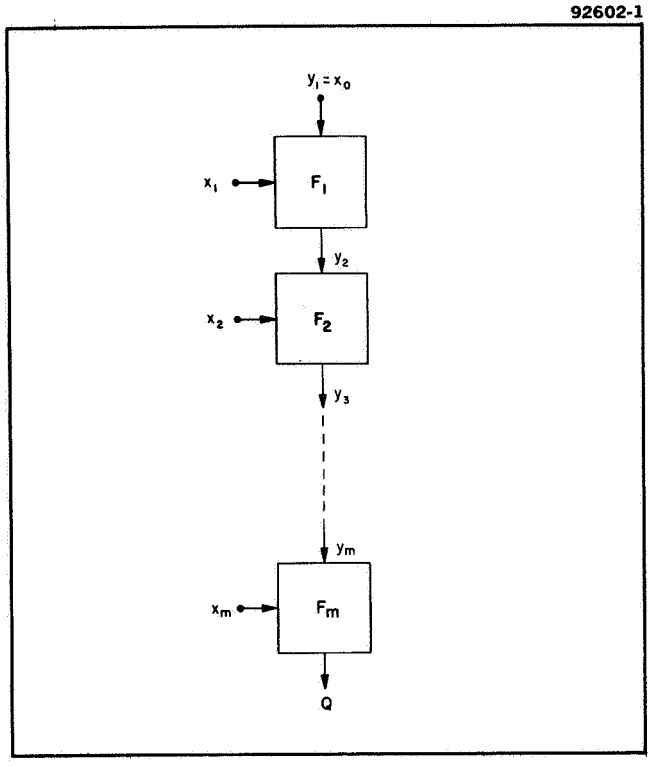


Figure 5-1. A Single Rail Cellular Cascade. The set of functions Q is smaller than the set of n-variable Boolean functions.

CHARACTERISTICS OF CUTPOINT CELLULAR ARRAYS

A single cutpoint cellular array can mechanize more than a single function. By rotating the bottom column of an array 90°, arbitrary functions of variables can be formed as the sum of products.

Minnick (Refs. 6 and 7) has devised single rail cellular arrays which he calls cutpoint cellular arrays. The binary cells which make up these arrays are called cutpoint cells because of Minnick's suggested method of cutting prewired connections to form the binary cell functions. The single rail cellular array is an extension to two dimensions of the one-dimensional cascade of Maitra. A single rail cellular array (cutpoint array) is shown in Figure 5-3. The horizontal inputs to each cell in the same row are identical. The indices in each cell denote the function to which the cell is set. The cell may be set to one of eight Boolean functions of two variables or set to function as an R-S flip-chip as shown in Figure 5-2.

As indicated in Figure 5-3, a single array can mechanize more than a single function. In the example,

$$F_1 = X_1X_2' + X_1'X_2 + X_3' + X_4$$

$$F_2 = X_2X_4$$

$$F_3 = X_3X_4$$

Minnick has shown (Refs. 6 and 7) that arbitrary functions of variables can be formed as the sum of products (product of sums). This method requires one column of cells for each product (sum) as well as a column which is rotated 90° to form the bottom row of the array to "collect" the products (sums). An example of a three variable function which forms the sum of products:

$$F = X_1'X_3 + X_2'X_3 + X_1X_2X_3'$$

is given in Figure 5-4. Intermediate Boolean Variables are shown at the output of each cell.

One set of rules (not the only one) for mechanizing combinational logic functions by "collecting" products is as follows:

1) Place independent variables along the left hand inputs.
2) Label each upper input to the rotated bottom row cells with one of the products.
3) Place a "1" in every cell for which the row variable is not an implicant in the product.
4) Place a "6" in every unfilled row 1 cell.
5) Place a "4" in each cell of the rotated bottom row.
6) Place a "0" in the right hand input of the bottom row.
7) Enter a "2", "3", "4", or "5" in the remianing cells to satisfy each product term working from the bottom up.
8) The upper inputs to the top row cells must be either "1" or "0".

An analogous set of rules can be written for collecting sums.

92602-2

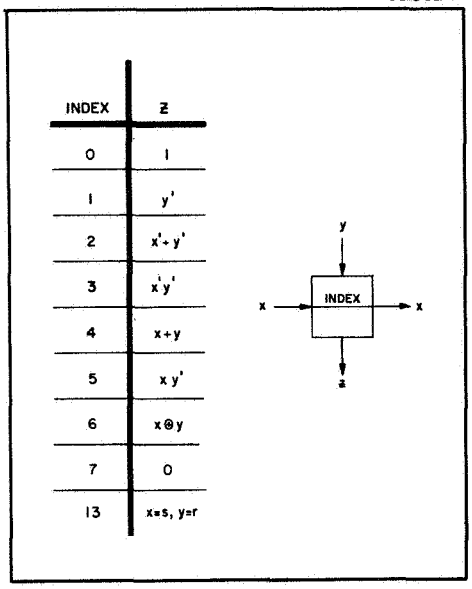


Figure 5-2. Functions for Cutpoint Cells. Cutpoint cells may either be set to one of 8 Boolean functions of 2 variables or may function as an R-S flip chip.

92602-3

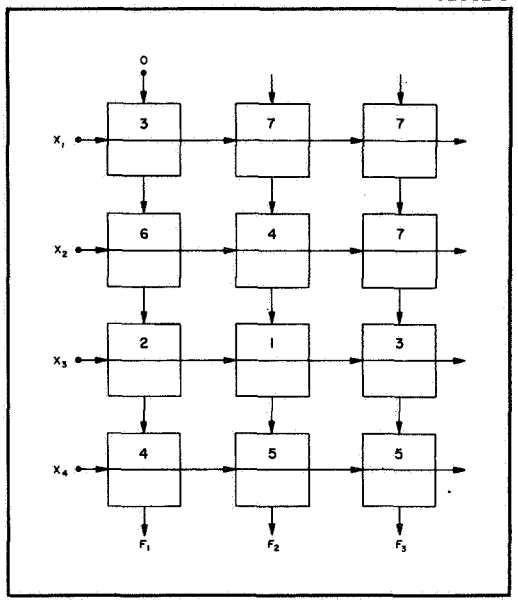


Figure 5-3. Example of a Cutpoint Array. This single rail cellular array allows for the mechanization of multiple functions.

92602-4

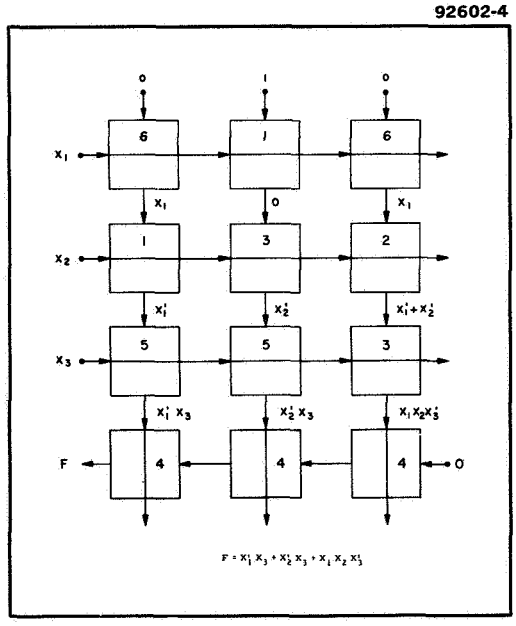


Figure 5-4. Example of a Three Variable Function which Forms the Sum of Products. One column is required for each product and one row to collect the products.

CHARACTERISTICS OF COBWEB CELLULAR ARRAYS

Although the functions implemented by a cobweb are the same as for a cutpoint array, a cobweb array provides a gain in efficiency because of fewer cells.

The cobweb cellular array is a single rail cellular array invented by Minnick (Ref. 8) to overcome four problems with cutpoint arrays:

- Inefficiency in the number of required cells.
- Excessive requirement for jumper connections from edge-output points to edge-input points of the same array.
- Insufficient number of edge connections to the array.
- The lack of cell isolation during early production so that step-and-repeat testing could be used to identify faulty cells.

The functions implemented by each cobweb cell remain the same as for the cutpoint cell. However, the interconnection complexity of the cells is increased by permitting inputs to each cell from two of five places: (1) the cell above, (2) the cell to the right, (3) the cell a knight's move above and to the right, (4) a horizontal feed-through bus, and (5) a vertical feed-through bus. The cell is shown in Figure 5-5.

In order to distinguish cobweb cell inputs, a circle is used for the x input, while a triangle is used for the y input. Jumper connections may be made to bypass a cell. A jumpered cell is indicated by placing a J in the cell and circling the jumpered connections. If more than one pair of connections is jumpered, a triangle is used to indicate the second pair of jumpered connections. A comparison of Figures 5-6 and 5-7 indicates the relative gain in efficiency to be made over a cutpoint array. The cobweb array uses 30 instead of 49 cells for the three-bit parallel adder. Another improvement is the carry chain which is reduced from 8 inverters to 6 inverters (assuming Minnick's DTL mechanization).

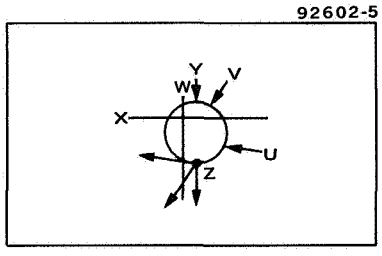


Figure 5-5. The Cell for Cobweb Cellular Arrays. Four major problems of the cutpoint array are overcome by the increased connectability of the cobweb cell.

92816-36

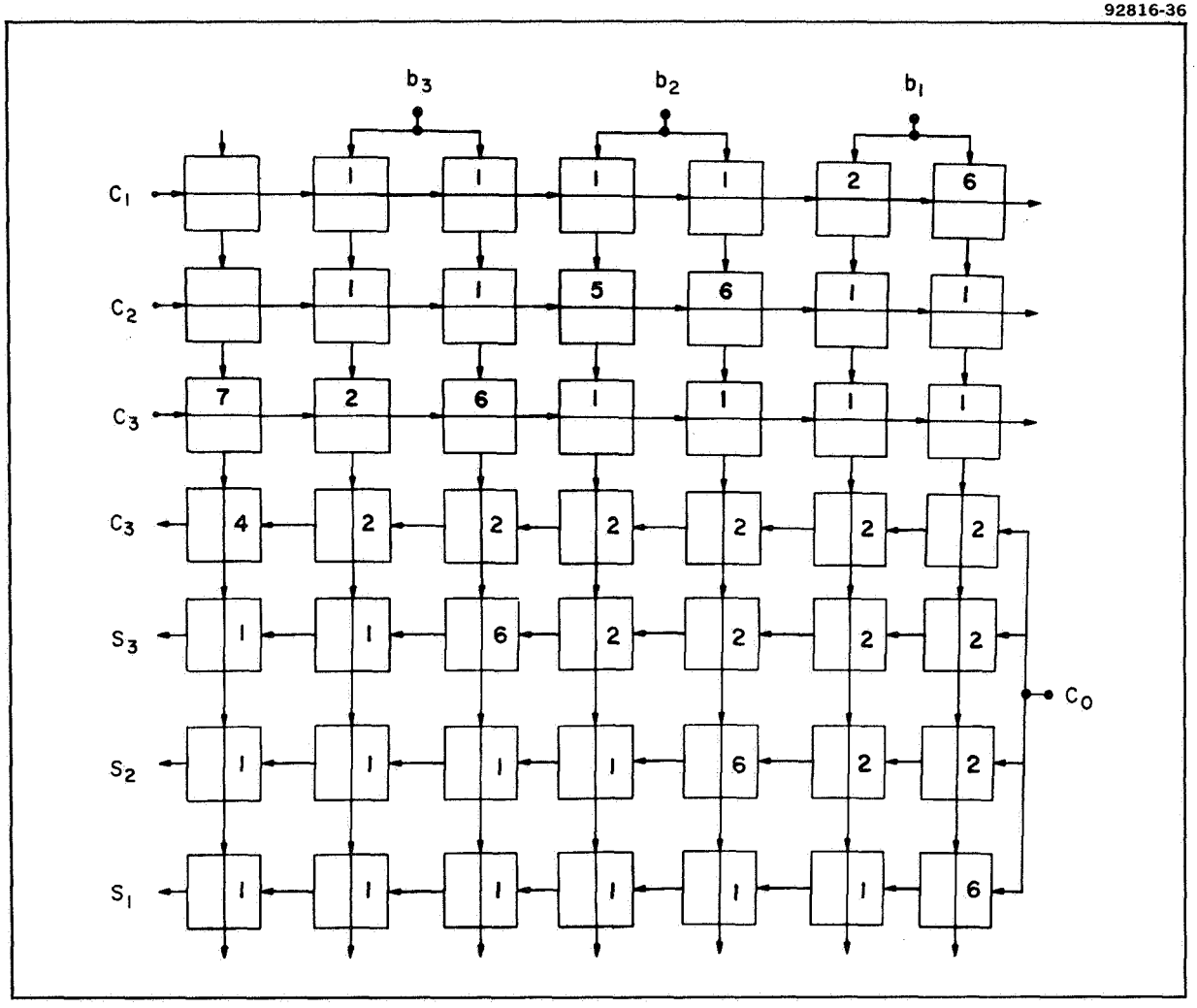


Figure 5-6. Cutpoint Array Realization

92816-37

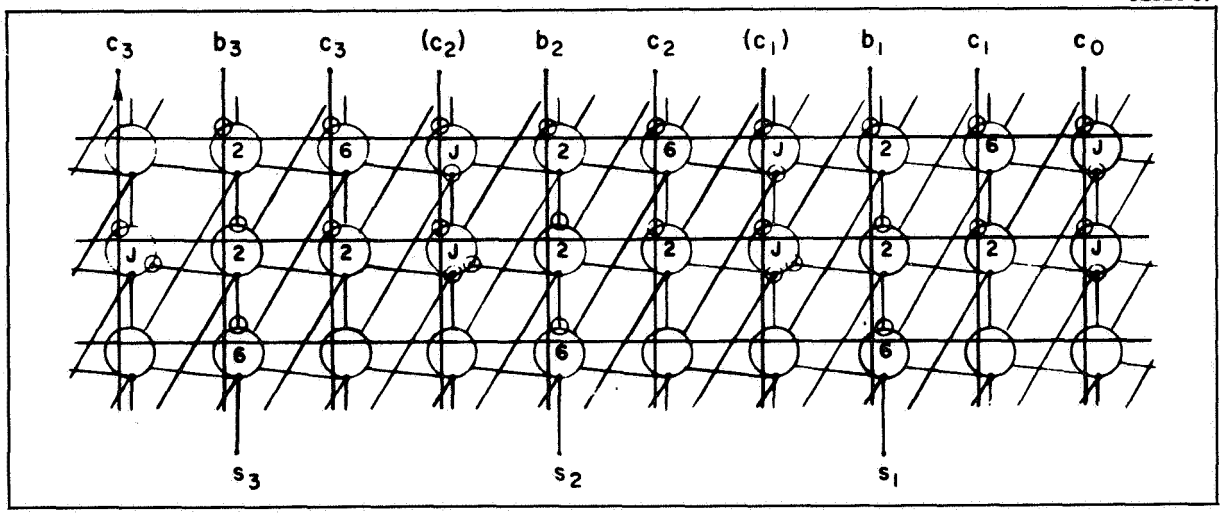


Figure 5-7. Cobweb Array Realization. Fewer cells are required to implement the same function.

COMPARISON OF CELLULAR AND MASTER SLICE MECHANIZATIONS

In terms of propagation delays and numbers of cells, cellular arrays offer no advantages over the commonly used master slice mechanization. Further study of Short's approach for two-rail cellular cascades is suggested.

Methodological Approach – The P2 functional character was chosen to be implemented with cellular logic. A preliminary implementation of the P2 functional character requires two types of flip-flops. The first is a trigger type, T, with asynchronous set-reset inputs and a clock input. The second flip-flop is a delay type, D. For the sake of comparing a cellular versus a master slice mechanization, it is assumed that either of these flip-flops is available when index thirteen is specified in the cellular mechanization. The comparison then reduces to one for combinational logic arrays. Since each cell in the cellular array is roughly equivalent in complexity to a gate in a typical T^2L master slice array, counting cells in a cellular array and gates in the master slice array gives the means for assessing the relative efficiency of the two mechanizations. This also gives an indication of the relative power dissipations of the two mechanizations. The number of propagation delays associated with the combinational array is a second factor to be considered.

Logic for the Chosen Functional Character – Typical of the logic for the P2 character is the count logic for the up-down counter. In the following Boolean equations. $Q_n{}^T$ is the trigger input to the nth bit position flip-flop in the counter. γ is the last carry out of the counter. This set of equations is mechanized in three ways: First, γ is mechanized with cutpoint cells, then the entire set of equations is mechanized with cobweb cells, and finally, the array is mechanized with NAND gates from a master slice.

$$Q_5{}^T = \phi_2\ \alpha_1\ Q_4 + \phi_2\ \alpha_2\ Q_4$$

$$Q_6{}^T = \phi_2\ \alpha_1\ Q_4 Q_5 + \phi_2\ \alpha_2\ Q_4 Q_4$$

$$Q_7{}^T = \phi_2\ \alpha_1\ Q_4 Q_5 Q_6 + \phi_2\ \alpha_2 Q_2 Q_4 Q_5 Q_6$$

$$\gamma = \phi_2\ \alpha_1\ Q_4 Q_5 Q_6 Q_7 + \phi_2\ \alpha_2 Q_4 Q_5 Q_6 Q_7$$

Cellular Mechanizations – A cutpoint cellular mechanization for the last carry output γ, is shown in Figure 5-8 on the following pages. The array was formed using the set of rules given for forming the sum of products. Eleven cells are required with seven cell delays. A NAND gate mechanization where 5 input gates were available would require 13 gates and 4 gates delays.

Master Slice Mechanization – Typically, master slice arrays contain an assortment of NAND gates and flip-flops such that arbitrary functions may be economically mechanized. In the mechanization shown in Figure 5-10, it is assumed that 5, 3, and 1 input gates are available in the ratio of 2:3:2. At least one LSI master slice is available in approximately these ratios. The array then requires 20 gates with four gate delays.

Results of Comparison – As might be expected, the number of delays associated with the cellular versions is generally greater than for the master slice version. This arises because the cellular arrays are based on the decomposition of Boolean functions a single variable at a time.

Two-rail cellular cascades as described by Short (Ref. 9) may give some relief in the length of propagation chains, but the cell complexity increases significantly beyond that of a single NAND gate.

It appears that cutpoint and cobweb cellular arrays give no significant advantages over a master slice array with respect to numbers of cells and propagation delay. The cobweb cell requires two layers of interconnect which is no better than the master slice. In order to reduce propagation delays, it seems worthwhile to pursue Short's approach for two-rail cellular cascades where a subset of all possible 3 variable cellular functions is used to compose arrays of considerate flexibility.

COMPARISON OF CELLULAR AND MASTER SLICE MECHANIZATION (Continued)

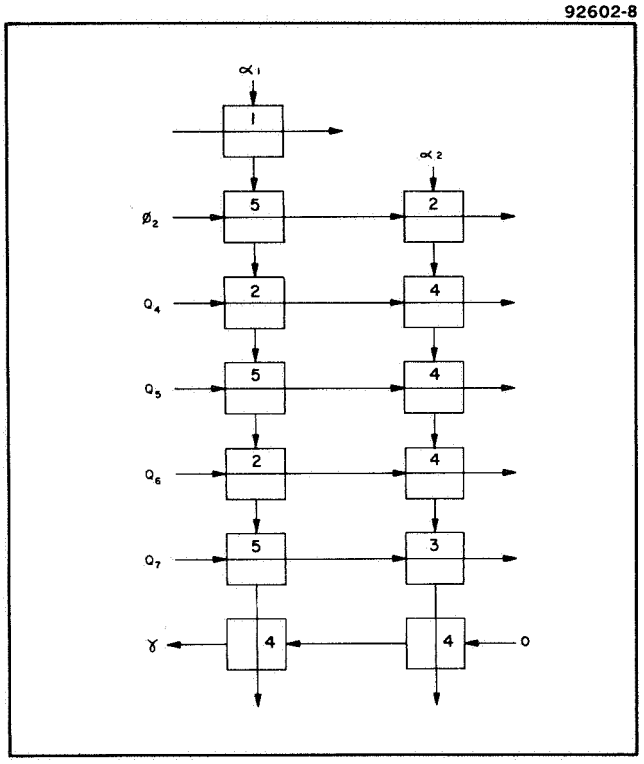


Figure 5-8. Cutpoint Mechanization for the Last Carry Output (γ) Logic. This mechanization requires seven gate delays, whereas a NAND gate mechanization requires only four delays.

92816-38

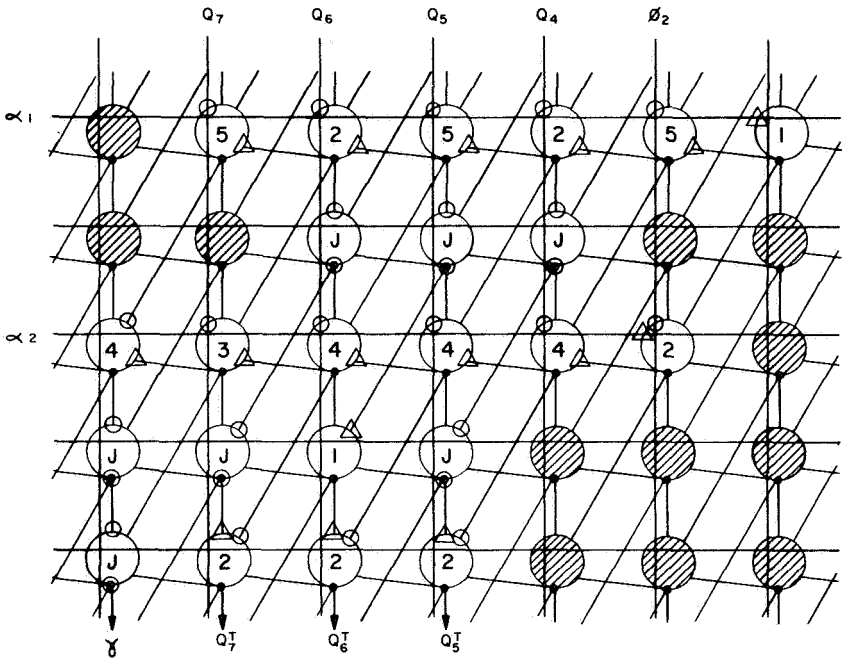


Figure 5-9. Cobweb Cell Array Mechanization for 4 Up-Down Counter Equations of P2 Character. Seven cell delays are required.

92816-39

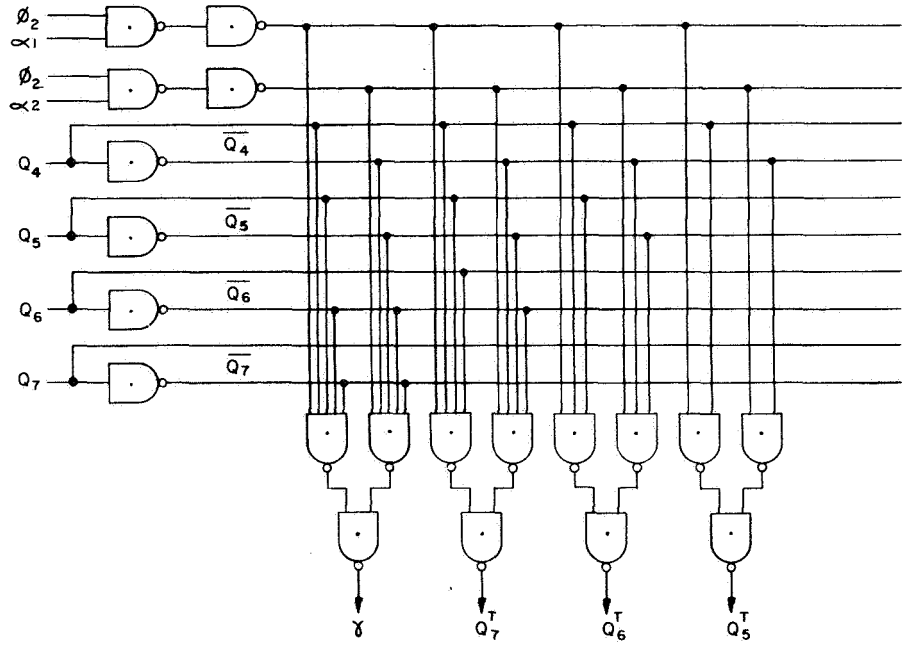


Figure 5-10. Master Slice Mechanization for 4 Up-Down Counter Equations of P2 Character. Four gate delays are required.

REFERENCES

(1) Minnick, R. C. A Survey of Microcellular Research, Journal of the Association for Computing Machinery, Vol. 14, No. 2, (April 1967), 203-241.

(2) Maitra, K. K. Cascaded switching networks of two-input flexible cells. IRE Trans. EC-11, 2 (April 1962), 136-143.

(3) Sklansky, J. General Synthesis of tributary switching networks. IEEE Trans. EC-12, 5 (Oct. 1963), 464-469.

(4) Levy, S. Winder, R. O., and Mott, T. H., Jr. A note on tributary switching networks. IEEE Trans. EC-13, 21 (April 1964) 148-151 (Correspondence).

(5) Stone, H. S. and Korenjak, A. J. Canonical form and synthesis of cellular cascades. IEEE Trans. EC-14, 6 (Dec. 1965), 852-862.

(6) Minnick, R. C. Cutpoint cellular logic. IEEE Trans. EC-13, 6 (Dec. 1964), 685-698.

(7) Minnick, R. C. Application of cellular logic to the design of monolithic digital systems. Proc. Symp. on Microelectronics and Large Systems, Spartan Books, Washington, D. C., 1965

(8) Minnick, R. C. Cobweb cellular arrays. Proc. AFIPS 1965 Fall Comput. Conf., Vol. 27, Pt. 1, pp. 327-341.

(9) Short, R. A. Two-rail cellular cascades. Proc. AFIPS 1965 Fall Joint Comput. Conf., Vol. 27, Pt. 1, pp. 355-369.

Section 6

Section 6 — Bibliography

The following comprise a basic set of reference material in the major areas of the study. Areas encompassed are aerospace computer characteristics, basic references on computer architecture, cellular logic, LSI applications, and earlier work in functional partitioning.

REFERENCES

- Avizienis, A., "Design of Fault-Tolerant Computers," AFIPS Conference Proceedings, Volume 31, Fall Joint Computer Conference, 1967.
- Baechler, D. O., "Trends in Aerospace Digital Computer Design," Computer Group News, Volume 2, No. 7, Pages 18-32, January 1969.
- Barnes, G. H., Brown, R. M., Kato, M., Kuck, D. J., Slotnick, D. L., and Stokes, R. A., "The ILLIAC IV Computer," IEEE Transaction on Computers, Volume C-17, Number 8, August 1968.
- Beelitz, H. R., Levy, S. Y., Linhardt, R. J., Miller, H. S., "System Architecture for Large-Scale Integration," AFIPS Conference Proceedings, Volume 31, Fall Joint Computer Conference 1967.
- Bersoff, E. H. and Hope, and F. Tung, IEEE Transactions on Aerospace and Electronic Systems, to be published.
- Bower, R. W. and Dill, H. G., "Insulated Gate Field-Effect Transistors Fabricated Using the Gate as a Source Drain Mask," International Electron Devices Meeting, Washington, October 1966.
- Bower, R. W., Dill, H. G., Aubuchon, K. G., and Tomps, S. A. "Characterization of MOS FETs formed by Gate Masked Ion Implantation," given at the International Electron Devices Meeting, Washington, D. C., October 1967.
- Cserhalmi, N., Lowenschuss, O., Scheff, B., "Efficient Partitioning for the Batch - Fabricated Fourth Generation Computer", Proceedings of the Fall Joint Computer Conference, 1968.
- Dickinson, M. M., Jackson, J. B., Randa, G. C., "Saturn V Launch Vehicle Digital Computer and Data Adapter," AFIPS Conference Proceedings 26, 501-516, Fall JCC 1964.
- Dill, H. G., "Offset Gate Field Effect Transistors with High Drain Breakdown Potential and Low Miller Feedback Capacitance," IEEE Transaction on Electron Devices, October 1968.
- "A Study of Jupiter Fly-by-Missions," General Dynamics Rept. FZM-4625, pp. 3-159 to 3-202, May 17, 1966.
- Jennings, R. C., "Design and Fabrication of a General Purpose Airborne Computer Using LSI Arrays, "Digest - 1968 IEEE Computer Group Conference, June 1968.

- Levy, S. Winder, R. O., and Mott, T. H., Jr. A note on tributary switching networks. IEEE Trans. EC-13, 21 (April 1964) 148-151 (Correspondence).

- Maitra, K. K. Cascaded switching networks of two-input flexible cells. IRE Trans. EC-11, 2 (April 1962), 136-143.

- Maurer, H. E. and Ricci, R. C., "Horizons in Guidance Computer Component Technology," IEEE Transactions on Computers, Volume C-17, No. 7, July 1968.

- Minnick, R. C. Application of cellular logic to the design of monolithic digital systems. Proc. Symp. on Microelectronics and Large Systems, Spartan Books, Washington, D. C., 1965.

- Minnick, R. C. A Survey of Microcellular Research, Journal of the Association for Computing Machinery, Vol. 14, No. 2, (April 1967), 203-241.

- Minnick, R. C. Cobweb cellular arrays. Proc. AFIPS 1965 Fall Comput. Conf., Vol. 27, Pt. 1, pp. 327-341.

- Minnick, R. C. Cutpoint cellular logic. IEEE Trans. EC-13, 6 (Dec. 1964, 685-698.

- Pariser, J. J., "Connection Considerations with a View Toward Batch Fabrication", Proceedings of the National Symposium of the Impact of Batch Fabrication on Future Computers, p. 213, April 1965.

- Pariser, J. J., Erwin, F. D., McKevitt, J. F., Burke, J. A., Disparte, C. P., and Schardin, C. H., "Research in the Effective Implementation of Guidance Computers with Large Scale Arrays," First Interim Report, Submitted to NASA ERC, October 1968.

- Segal, J., "Speed/Power Chart for Digital IC's," The Electronic Engineer, June 1968.

- Short, R. A. Two-rail cellular cascades. Proc. AFIPS 1965 Fall Joint Comput. Conf., Vol. 27, Pt. 1, pp. 355-369.

- Sklansky, J. General Synthesis of tributary switching networks. IEEE Trans. EC-12, 5 (Oct. 1963), 464-469.

- Stone, H. S. and Korenjak, A. J. Canonical form and synthesis of cellular cascades. IEEE Trans. EC-14, 6 (Dec. 1965), 852-862.

- Van Hoode, G. R., TRW, "Evaluation of Experience with Microelectronic Integrated Circuits," No. 9990-6183-R000, May 1967.

APPENDIX

RESEARCH IN THE EFFECTIVE IMPLEMENTATION OF GUIDANCE COMPUTERS WITH LARGE SCALE INTEGRATION

1968 Midterm Report prepared by Hughes-Fullerton presented to NASA ERC

Appendix – Research in the Effective Implementation of Guidance Computers with Large Scale Integration

The following pages contain reductions of the charts which were used in the verbal Mid-term Report given by Hughes at NASA ERC in 1968. Their content summarizes much of the material of this report. The bulk of these are self-explanatory and are not further annotated here. However, a few necessary added comments follow.

Figure A-7 illustrates the basic motivation for this study. Till now, the most common design technique has been first to design the logic for the system and then to partition accordingly. The character set was derived in a different manner. A study of computer structures revealed that in general they were organized as shown in Figure A-7. Having broken computer structures into these functional blocks, specific characters were designed to accomplish functions shown.

This approach is particularly relevant to upcoming LSI technology. With the complexity of functions possible on a single LSI chip, it is probable that current design techniques will create a large number of highly specialized and expensive chips. The functional character approach represents a solution to this problem, offering a small number of generally applicable circuits.

Figure A-21 shows the structure of the Hughes-designed MCB. There are two numbers associated with each unit. The first is the total number of characters needed to implement the unit. The second is the number of different character types in the unit.

Figure A-27 illustrates the difference in the I/O procedure of the first and the present Hughes-designed MCB. The original design involved the CU to the extent of making I/O functions a three-unit process. The CU was used as a command generator and timing base. In the present version these functions are taken over by the I/O unit. The CU is used only to initialize the I/O unit. Also, the I/O is now provided with a 32-bit data interface.

Figure A-34 is based on semiconductor manufacturer's projections. The lower "available on market" scale is Hughes' projection of the actual avilability of the hardware. It is our opinion that LSI devices of 200 gates or more are still forseeable by the 1970-1971 period.

Conventional MOS is compared to bipolar TTL in Figure A-35. MOS offers lower power dissipation, greater circuit density, and at least equal MTBF. The usual fabrication of MOS requires two masks, one for the source and drain and one for the gate. Manufacturing tolerances require a certain amount of gate overlap, which causes significant parasitic capacitance. Consequently conventional MOS is relatively slower than bipolar TTL.

Figure A-37 summarizes some reliability data on conventional MOS. The important point is that MOS failures are not uniformly distributed in time. A high percentage of failures occur in the first 150 hours. Therefore MOS has a high potential reliability if a "burn-in" time is allowed to eliminate defective circuits.

MOS transistors can be implemented by ion implantation. The source and drain can be formed entirely by implantation, or can be diffused beyond a certain gap around the gate which is then filled in by implantation. Implantation is usually done through the gate oxide layer in order to maintain planar device properties. Since the metal gate is no longer present to protect the gate-drain region from surface impurities, an Si_3N_4 film is added.

Appendix – Research in the Effective Implementation of Guidance Computers with Large Scale Integration

The advantage of the IMOS (over regular MOS) most pertinent to this study is higher speed due to reduced gate capacitance. Other advantages are summarized in Figure A-38.

Another advantage of IMOS is illustrated in Figure A-39. If the implementation of the characters is done with conventional circuits, one is forced to accept one of the indicated points on the speed-power curve. However, a customed-designed IMOS approach allows the designer to pick any point within the shaded region.

Figure A-40 presents the improvements made in the second functional character implementation of the modular computer and a comparison of the improved parameters with the modular computer breadboard.

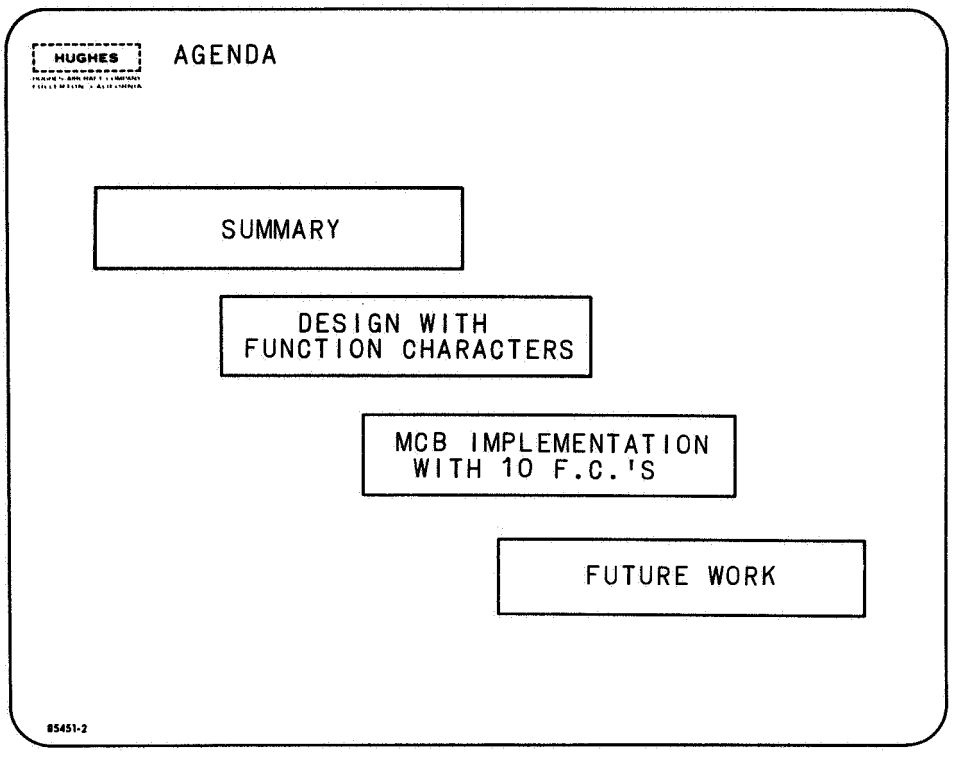


Figure A-1.

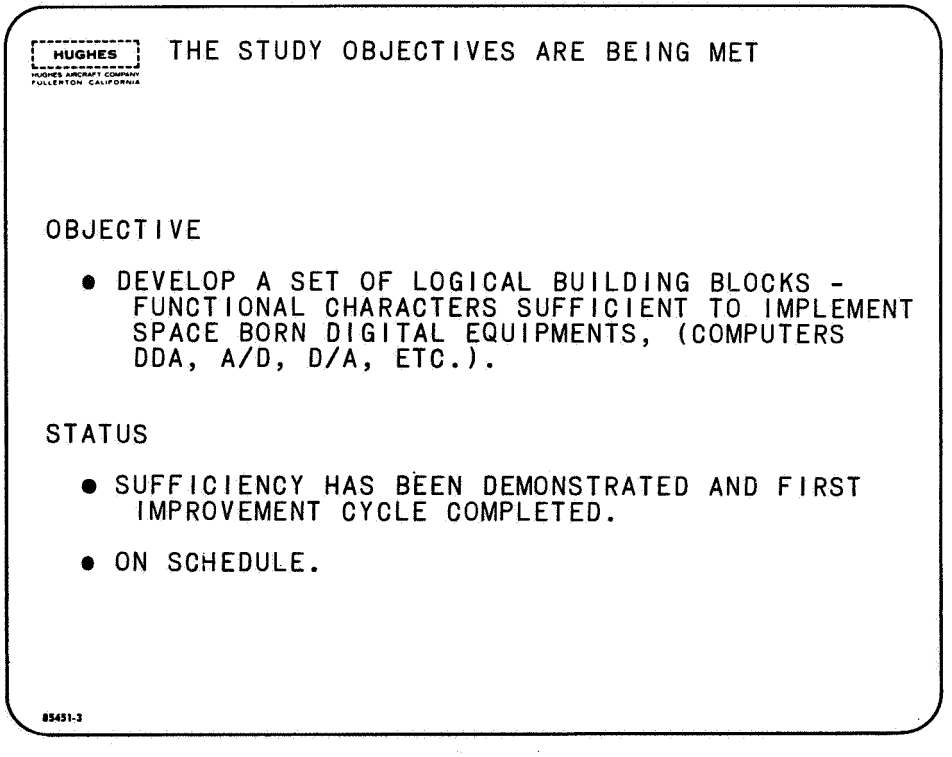


Figure A-2.

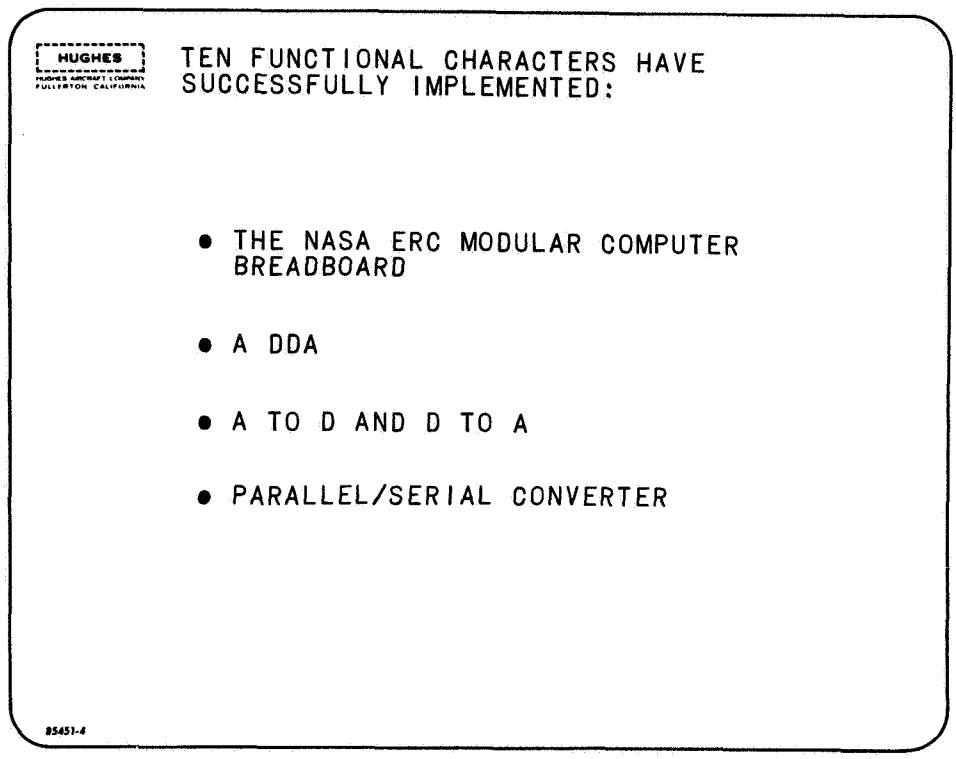


Figure A-3.

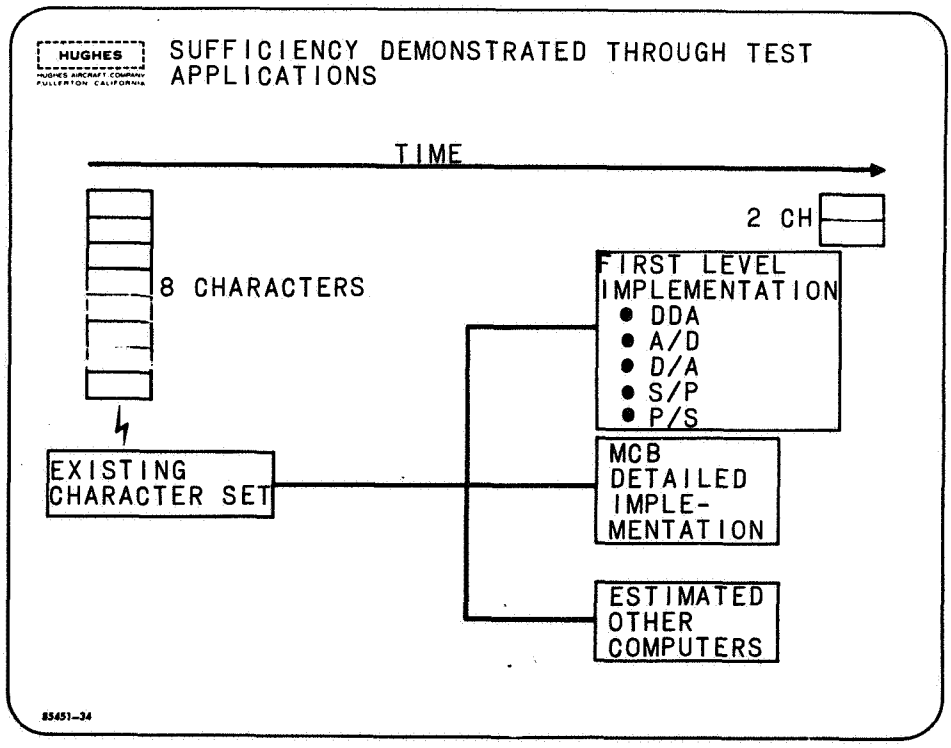


Figure A-4.

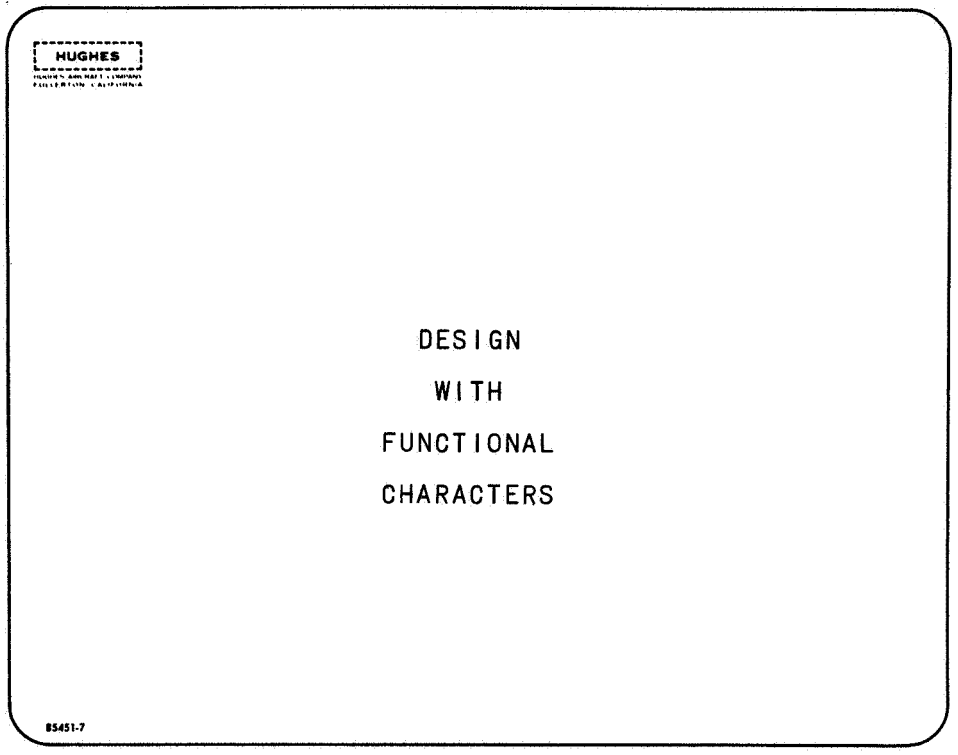


Figure A-5.

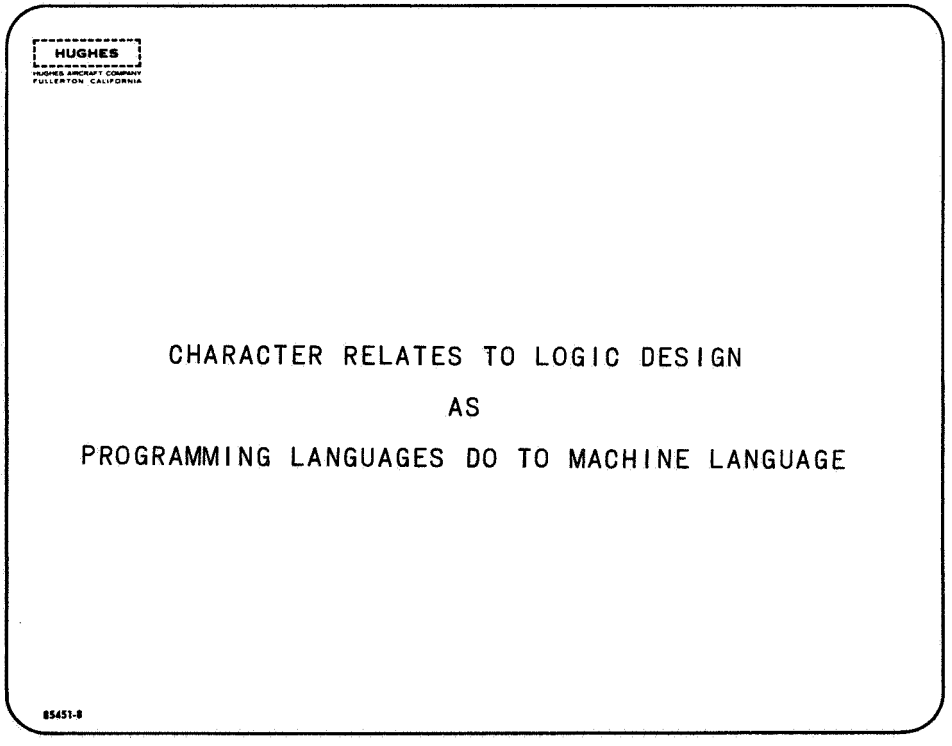


Figure A-6.

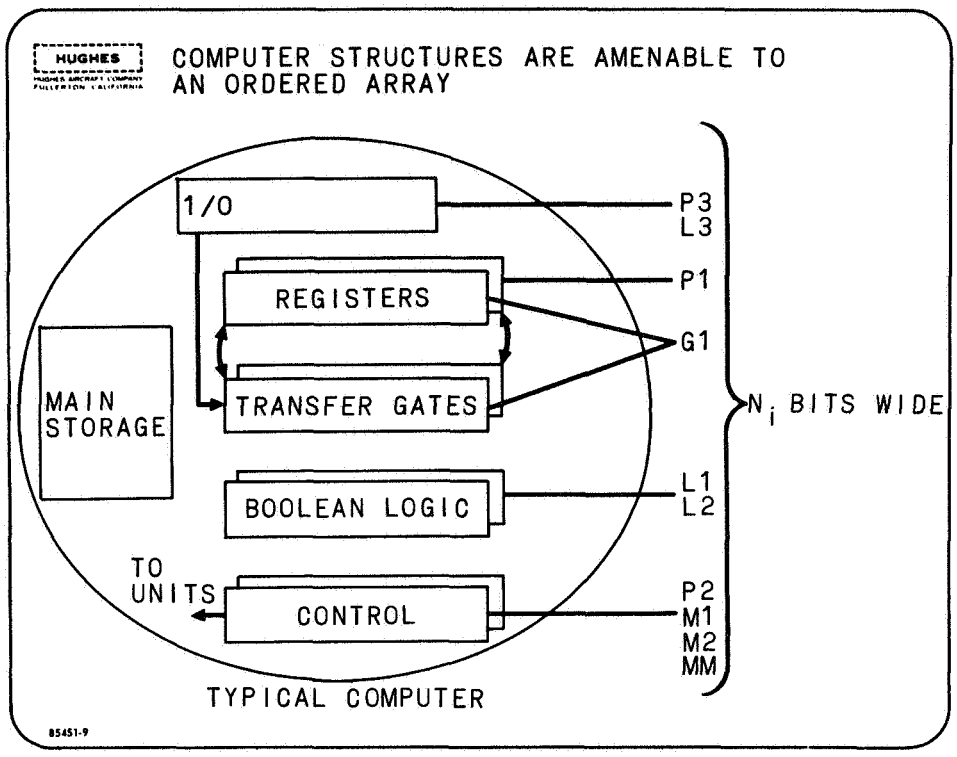


Figure A-7.

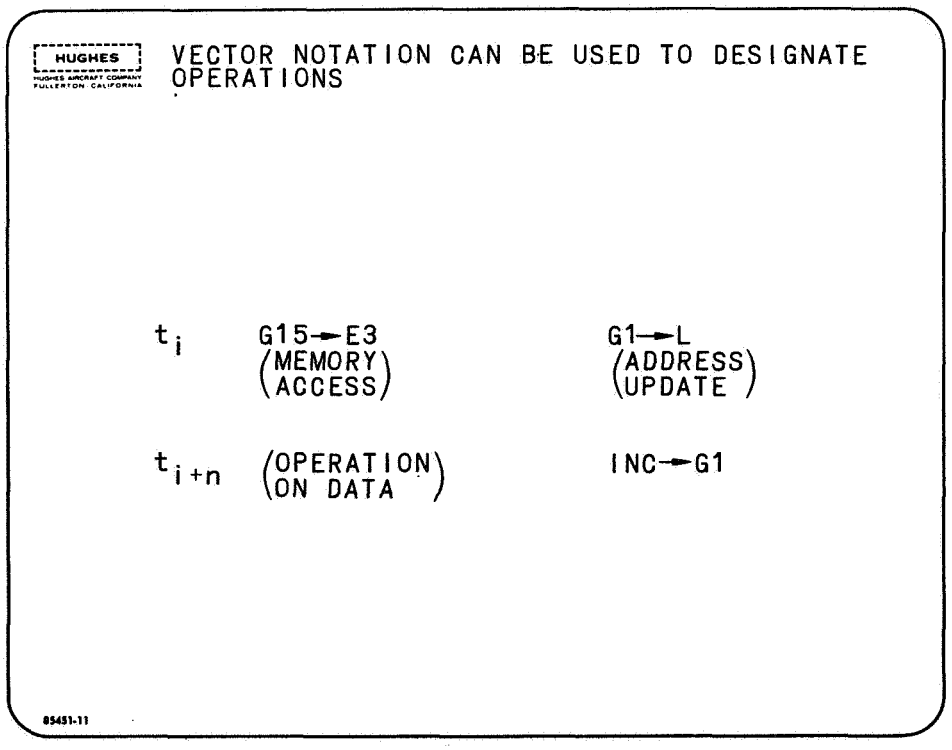


Figure A-8.

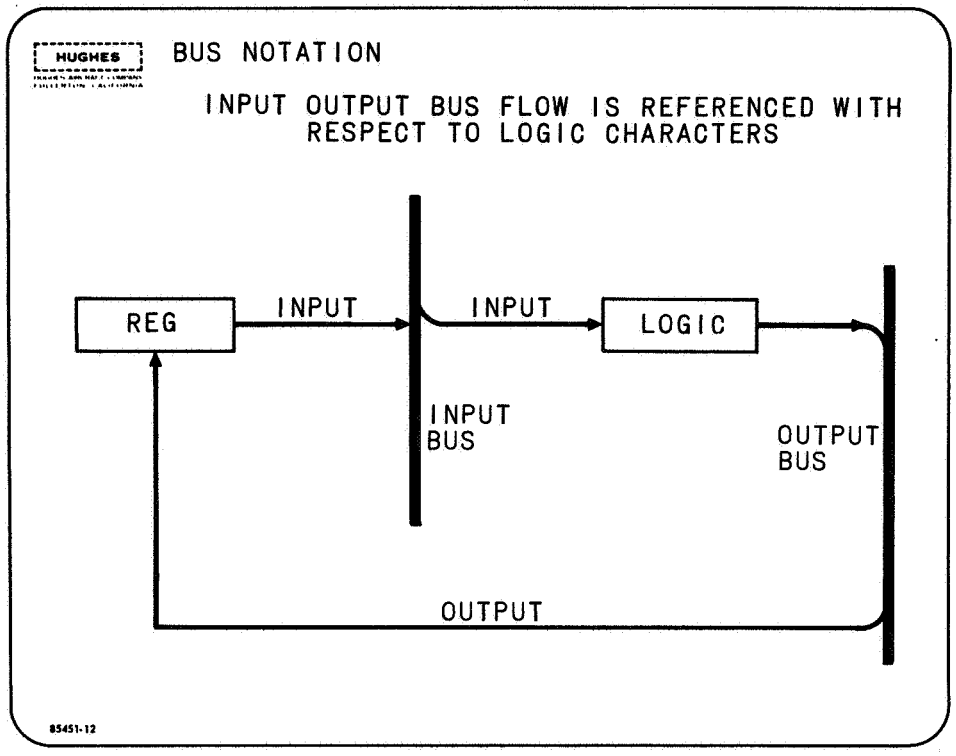


Figure A-9.

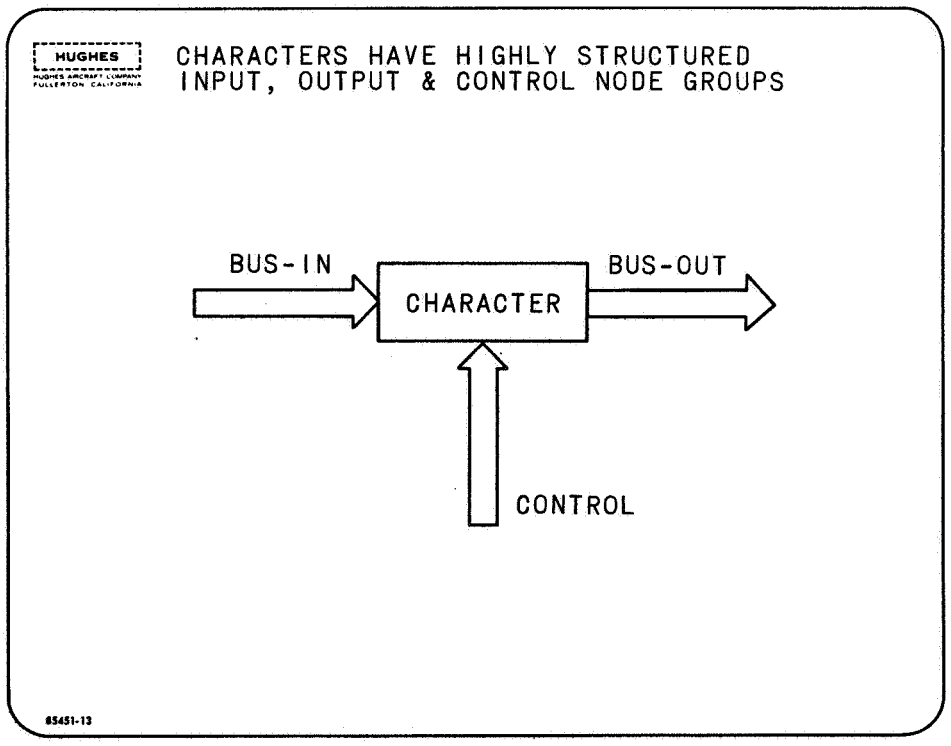


Figure A-10.

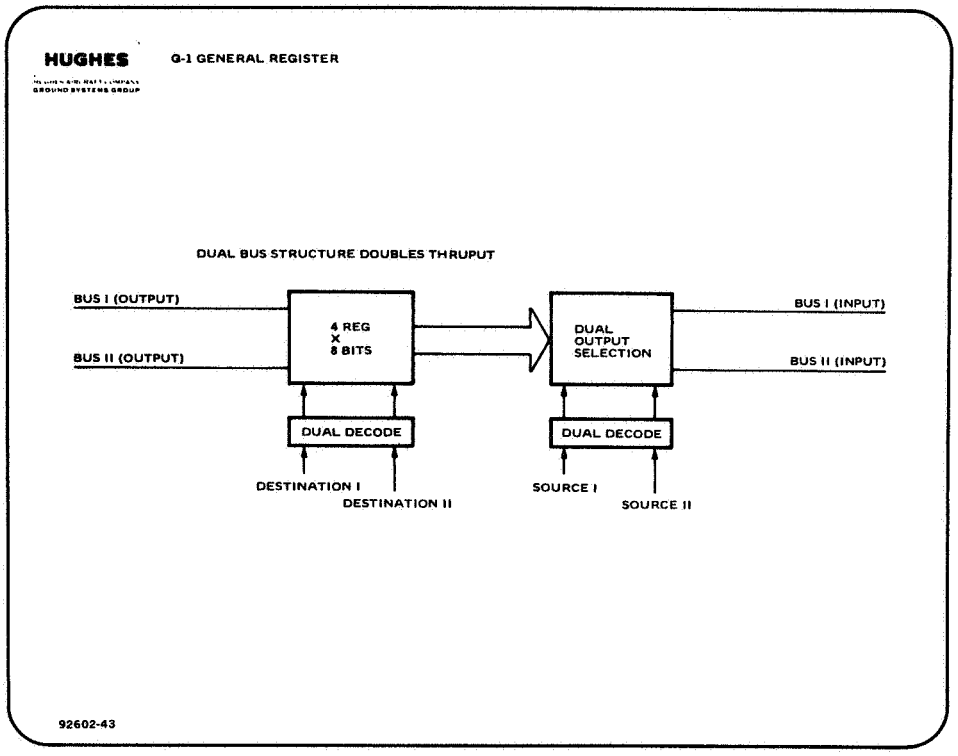


Figure A-11.

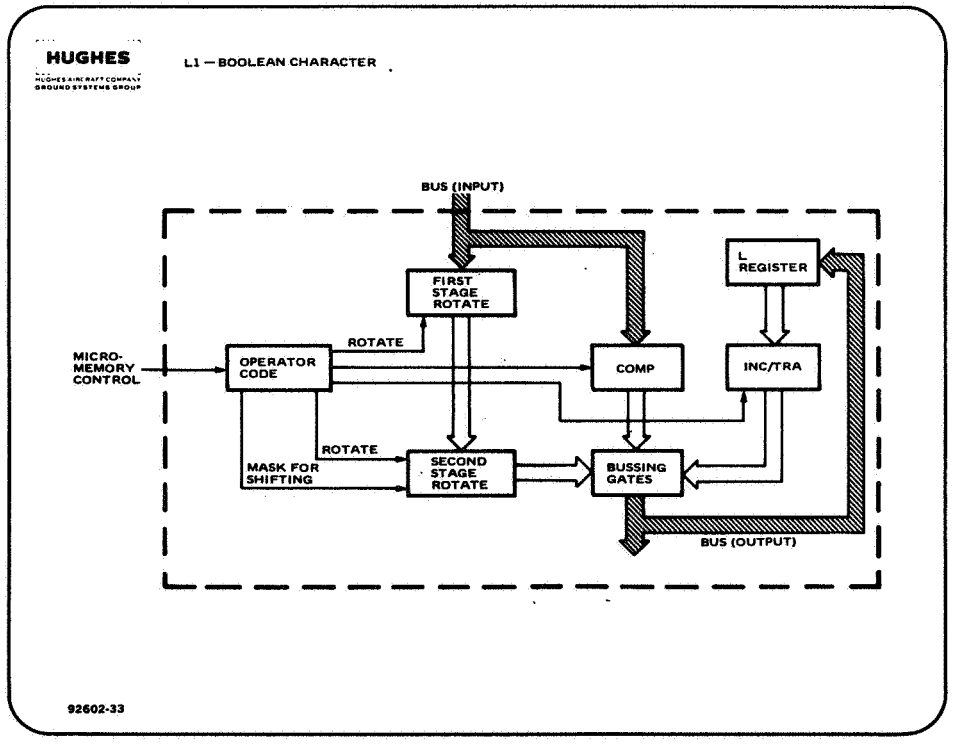


Figure A-12.

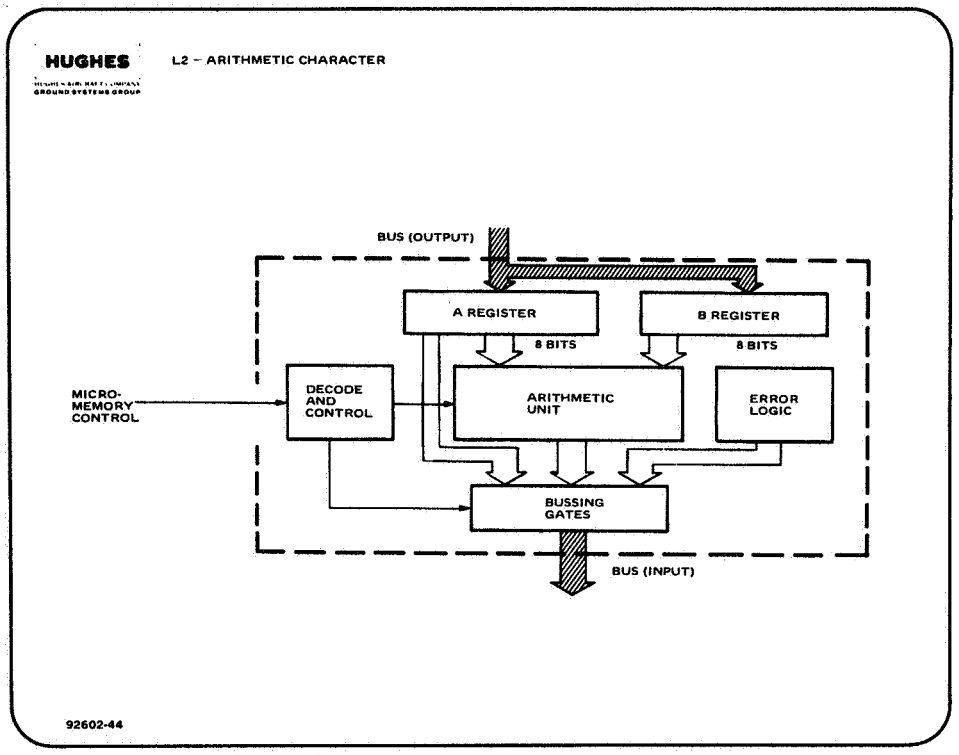


Figure A-13.

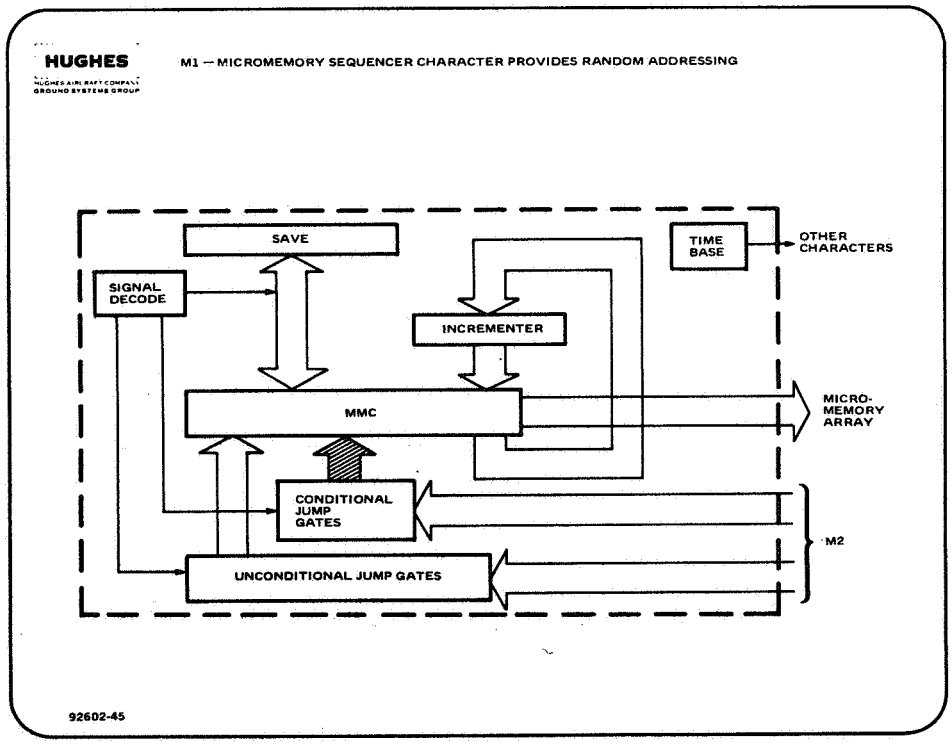


Figure A-14.

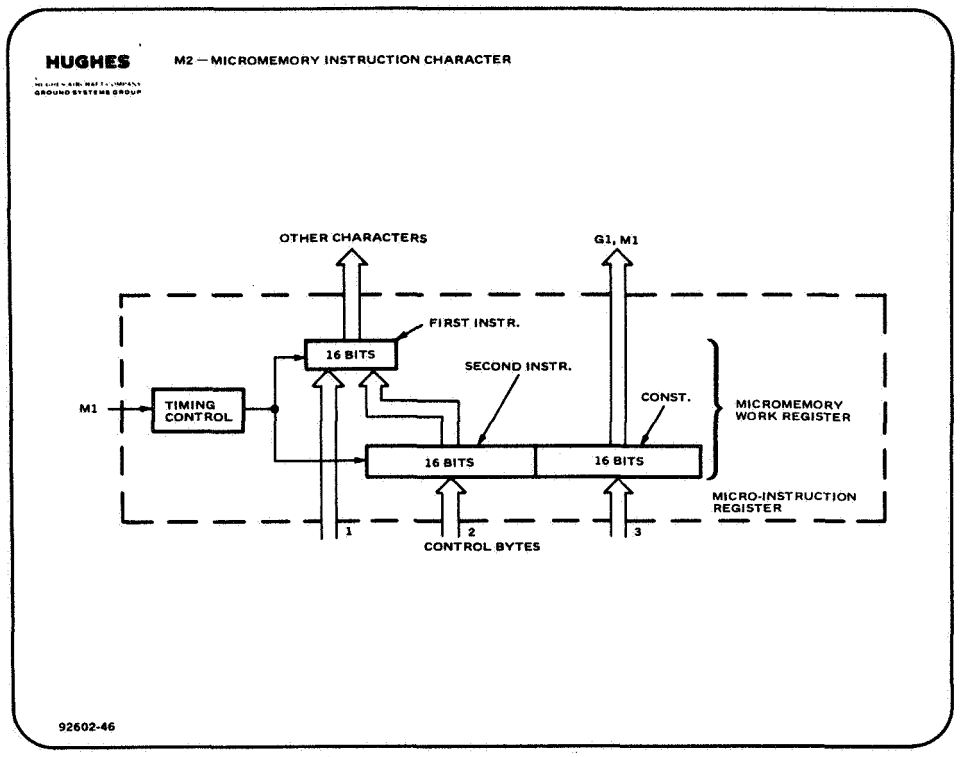


Figure A-15.

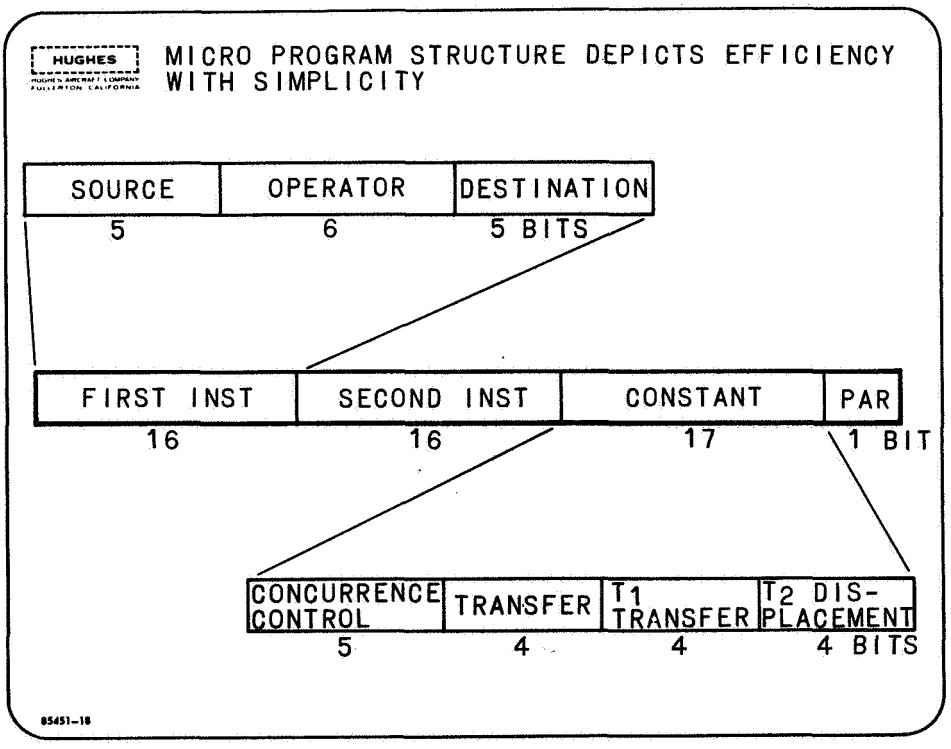


Figure A-16.

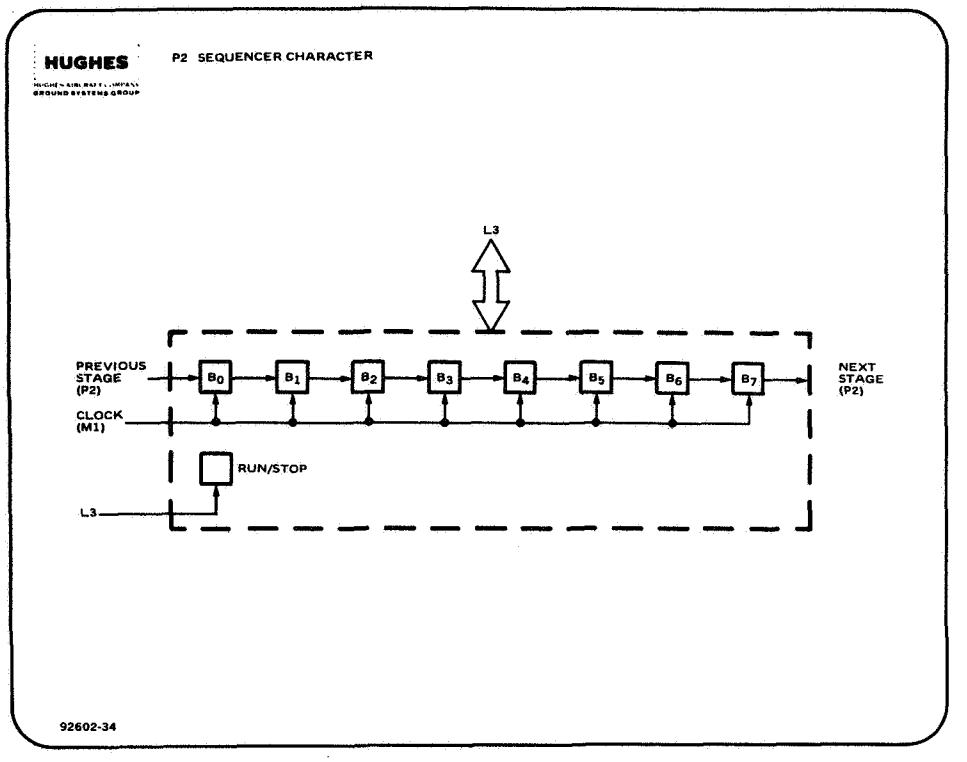


Figure A-17.

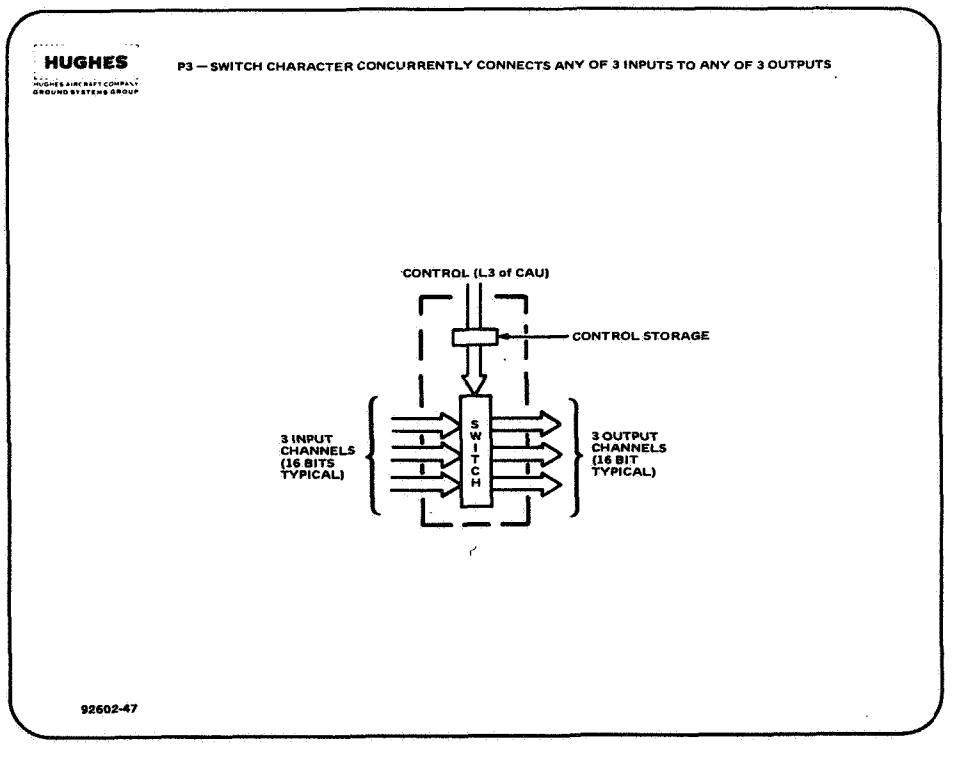


Figure A-18.

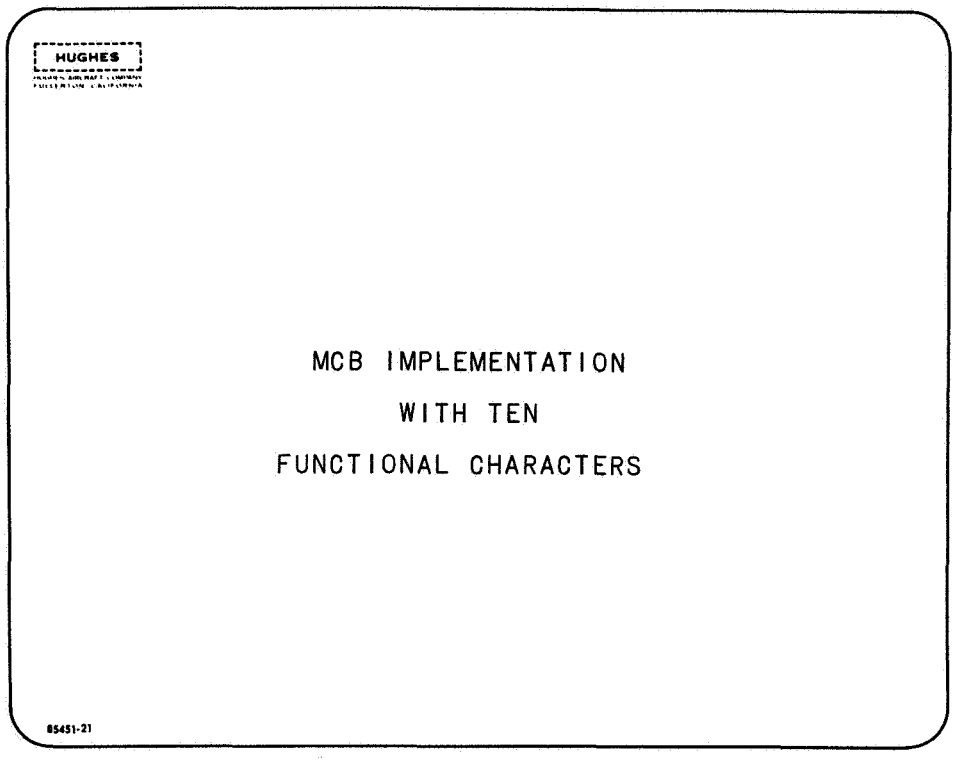


Figure A-19.

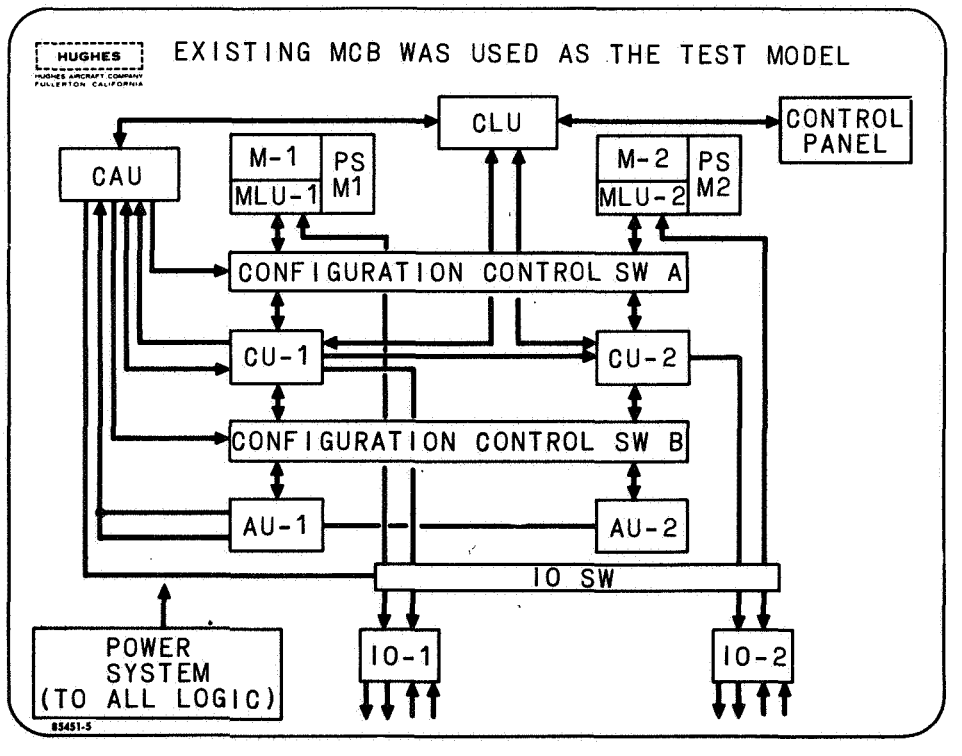


Figure A-20.

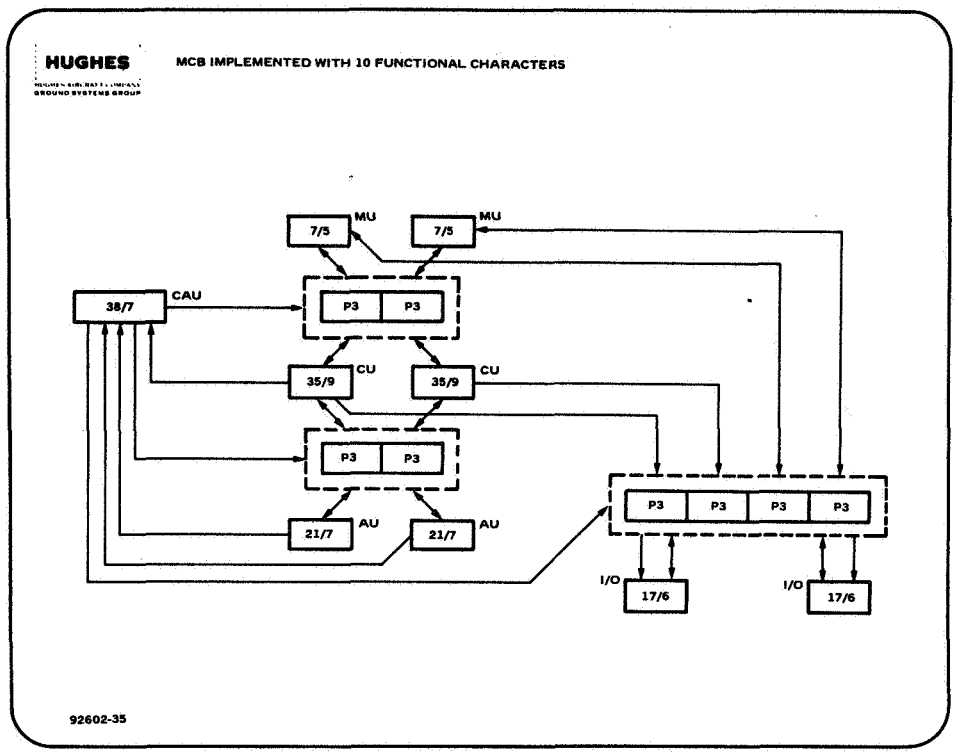


Figure A-21.

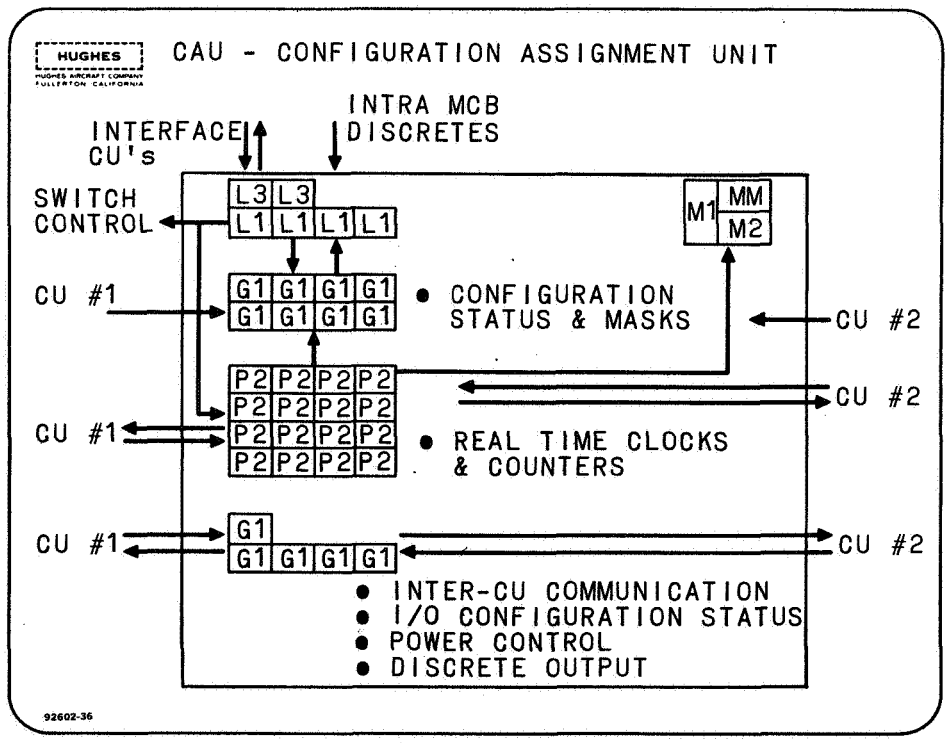


Figure A-22.

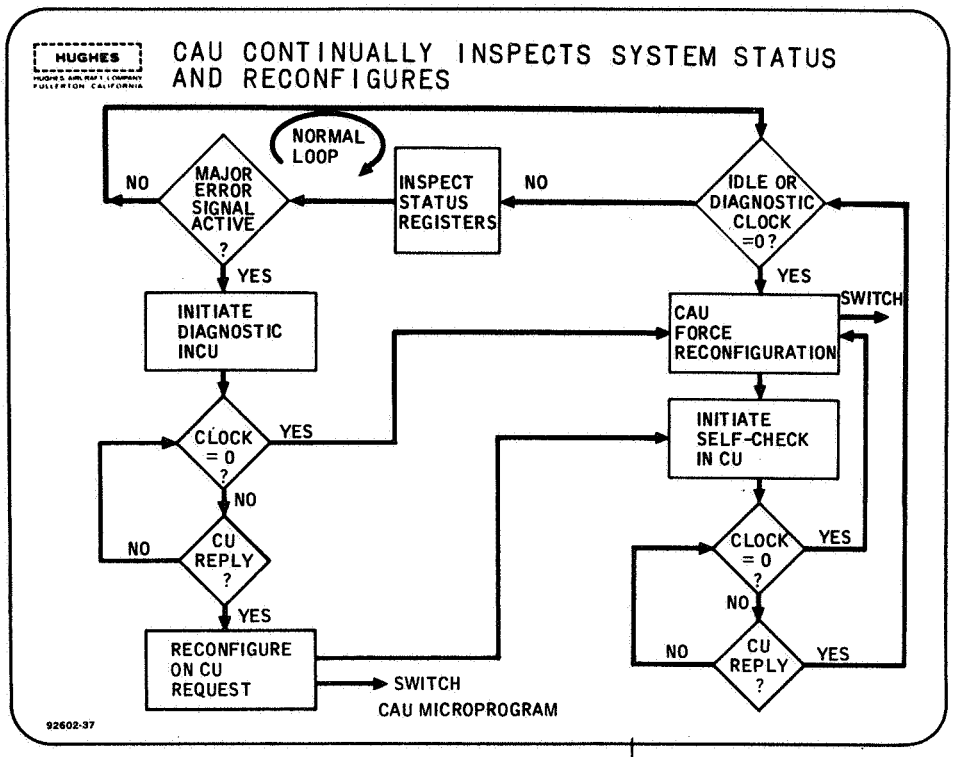


Figure A-23.

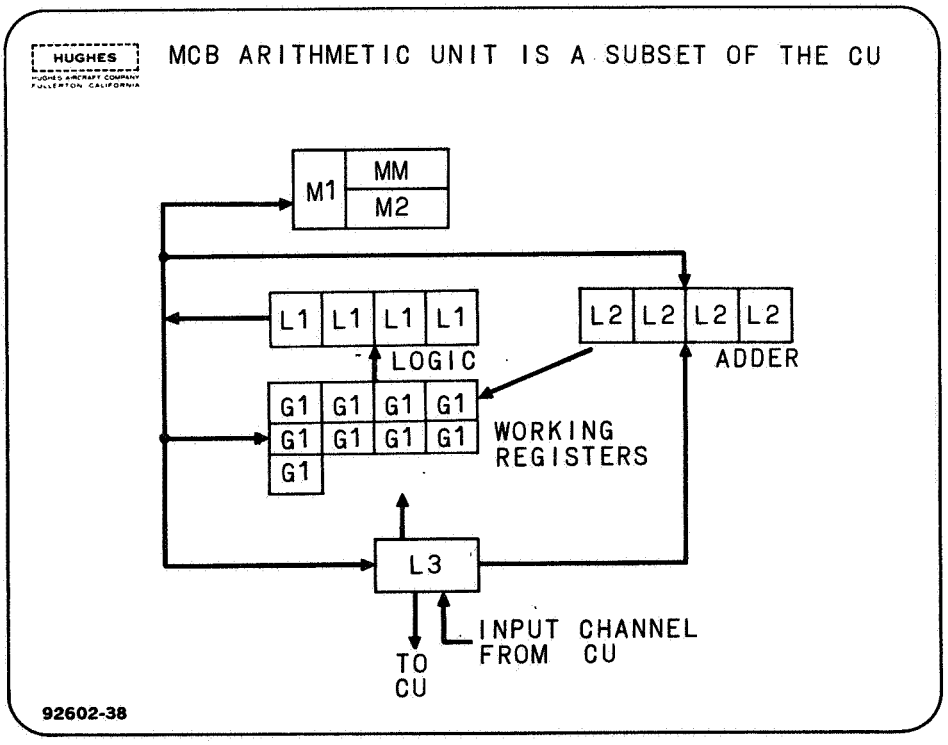


Figure A-24.

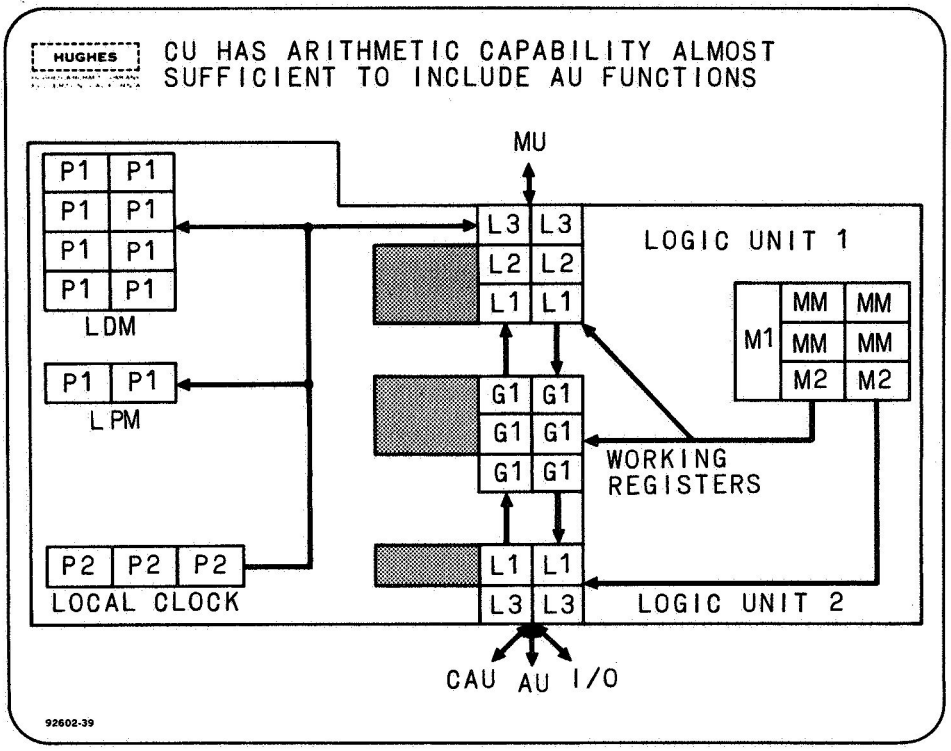


Figure A-25.

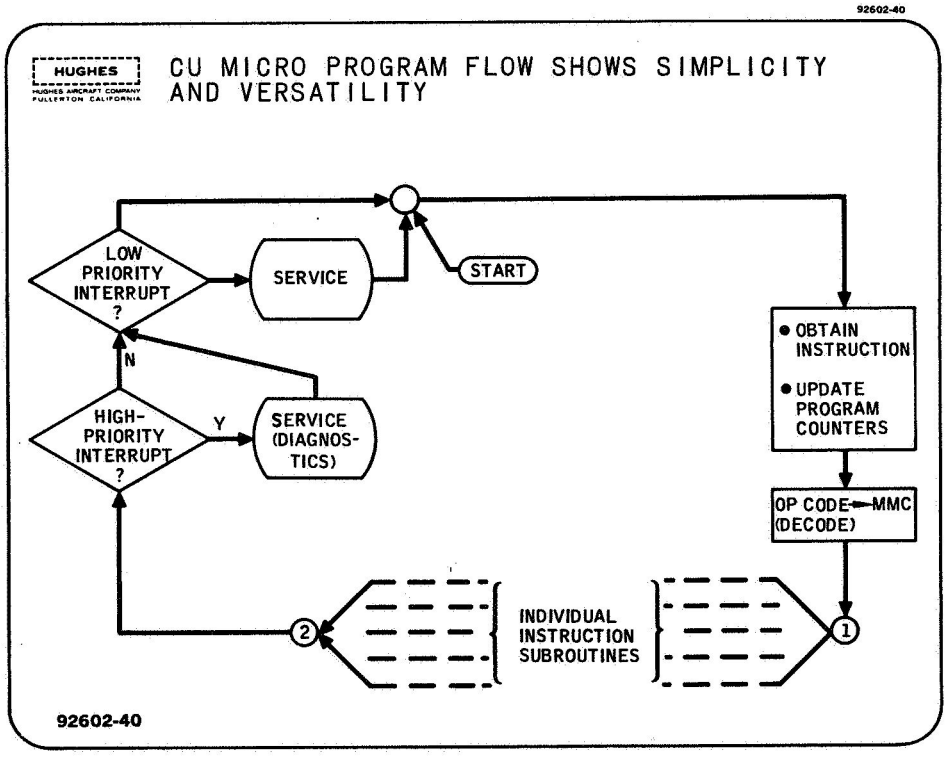


Figure A-26.

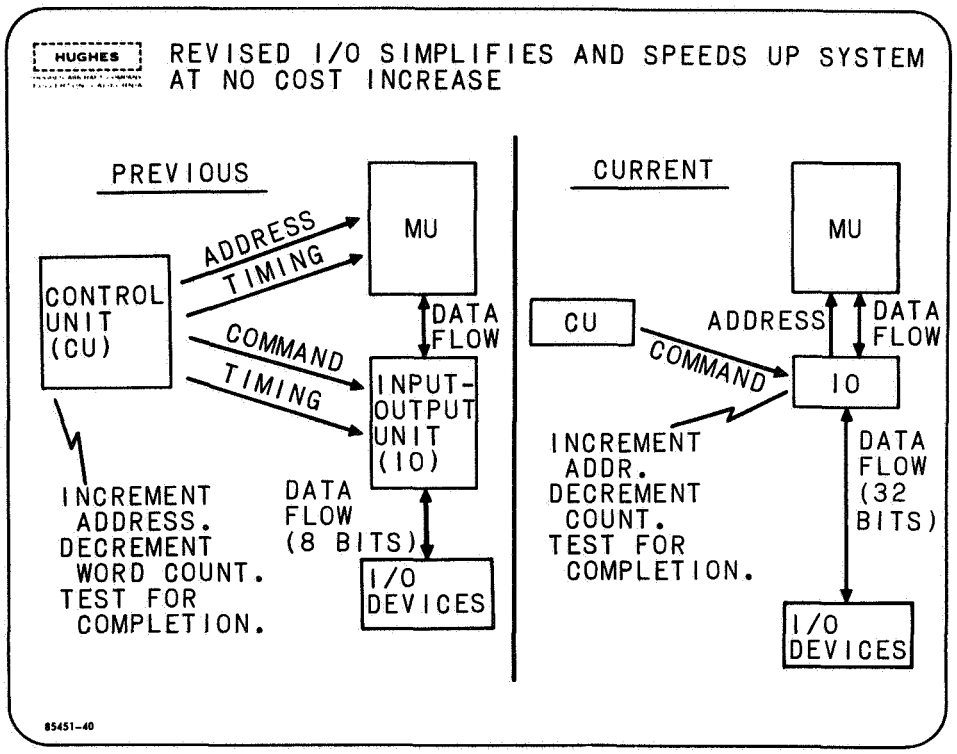
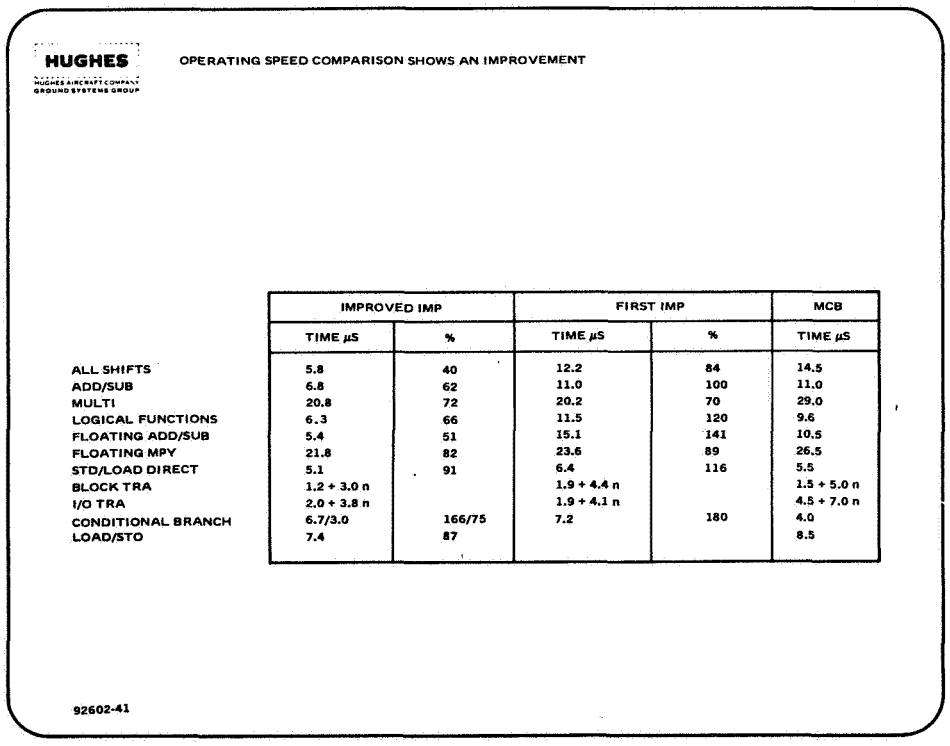


Figure A-27.



	IMPROVED IMP		FIRST IMP		MCB
	TIME μS	%	TIME μS	%	TIME μS
ALL SHIFTS	5.8	40	12.2	84	14.5
ADD/SUB	6.8	62	11.0	100	11.0
MULTI	20.8	72	20.2	70	29.0
LOGICAL FUNCTIONS	6.3	66	11.5	120	9.6
FLOATING ADD/SUB	5.4	51	15.1	141	10.5
FLOATING MPY	21.8	82	23.6	89	26.5
STD/LOAD DIRECT	5.1	91	6.4	116	5.5
BLOCK TRA	1.2 + 3.0 n		1.9 + 4.4 n		1.5 + 5.0 n
I/O TRA	2.0 + 3.8 n		1.9 + 4.1 n		4.5 + 7.0 n
CONDITIONAL BRANCH	6.7/3.0	166/75	7.2	180	4.0
LOAD/STO	7.4	87			8.5

Figure A-28.

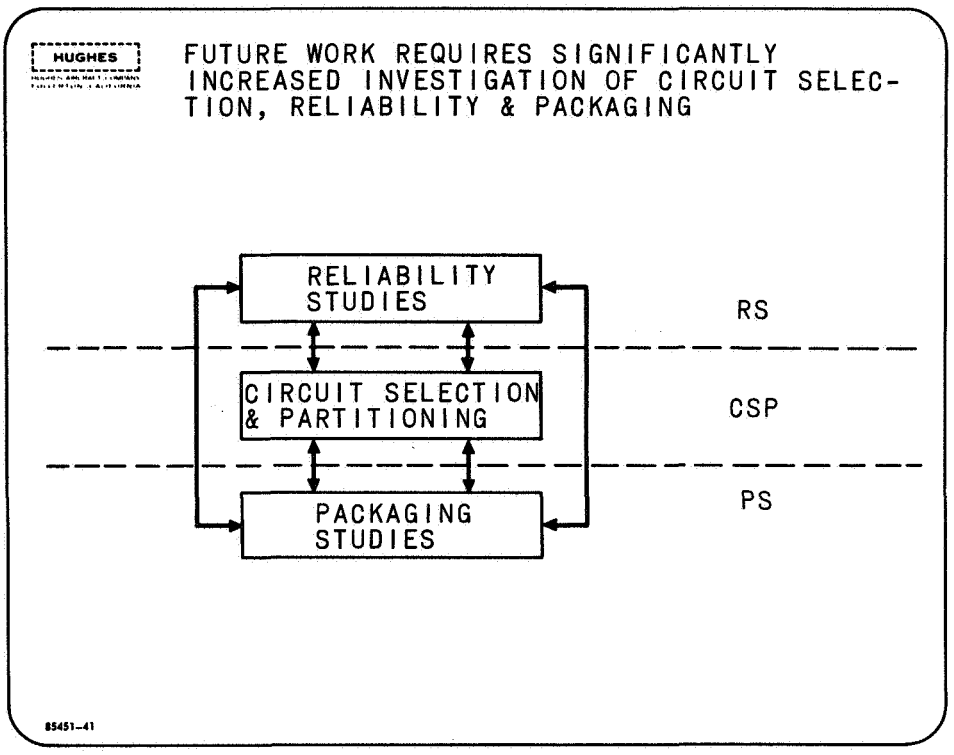


Figure A-29

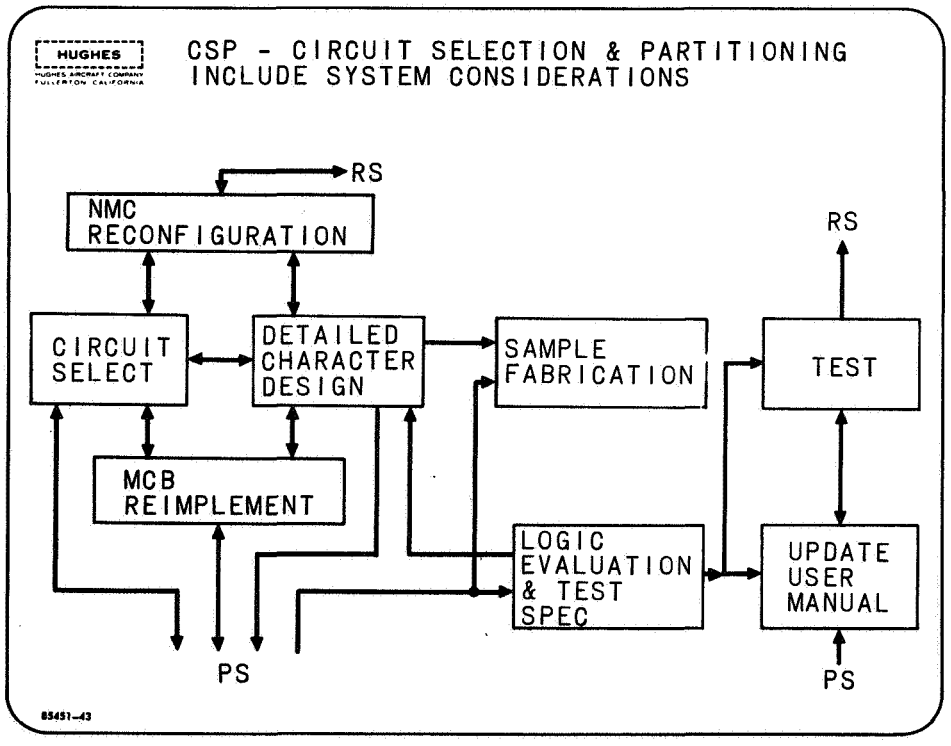


Figure A-30.

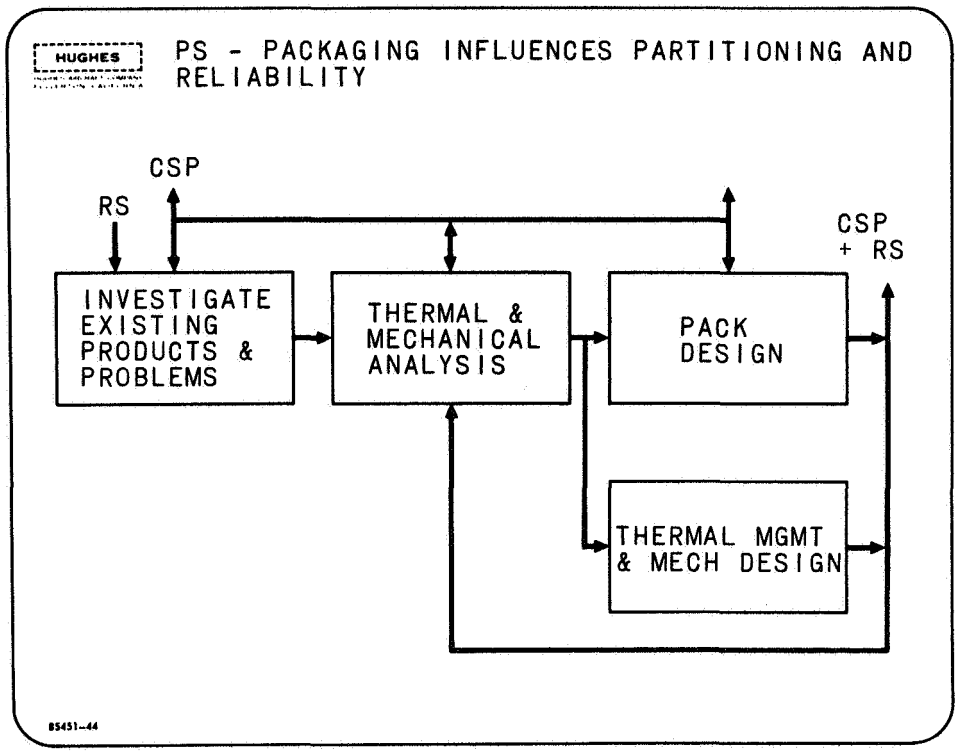


Figure A-31.

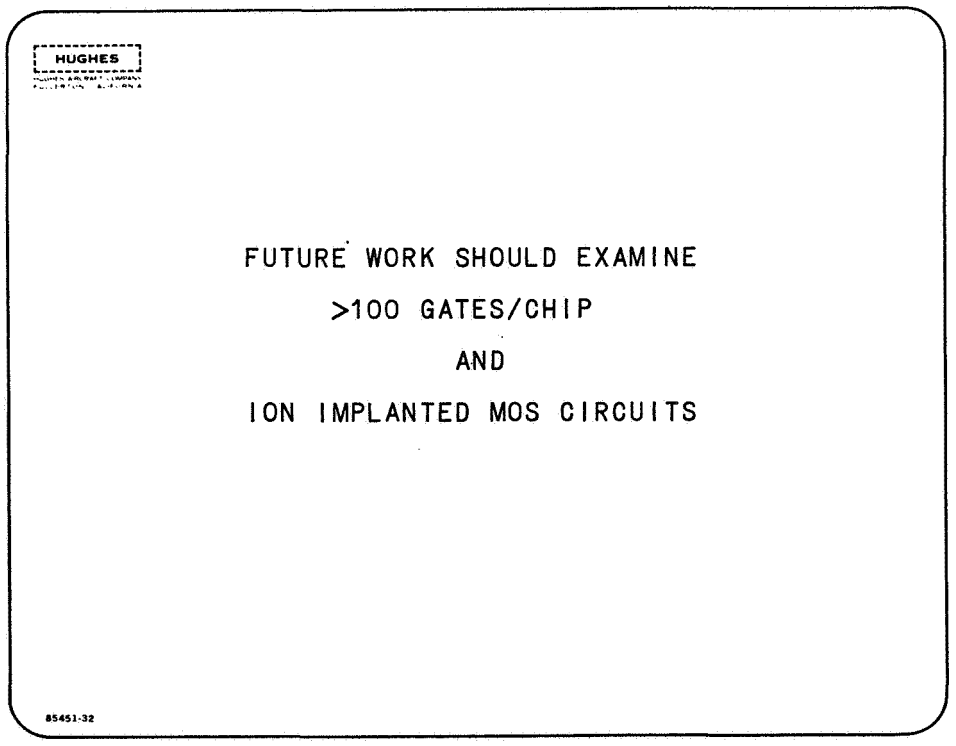


Figure A-32.

HUGHES

LARGER NUMBER OF GATES PER CHIP HAVE IMPROVED SYSTEM EFFECTIVENESS POTENTIAL

ADVANTAGES

1. GREATER GATES/PIN - HIGHER MTBF/GATE
2. LOWER POWER DISSIPATION/GATE
3. REDUCED NO. OF CHIP-TYPES (FUNCTIONAL)
4. REDUCED LABOR & PACKS

DISADVANTAGES

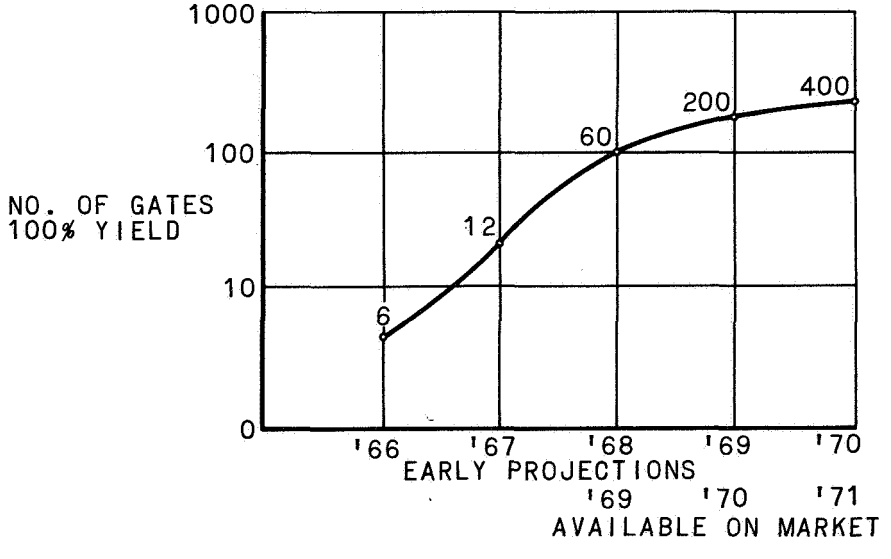
1. HARDER TO PRODUCE (LOWER YIELD)
2. INCREASED THERMAL CONCENTRATION
3. COULD LEAD TO MANY CHIP TYPES (NON-FUNCTIONAL)
4. COULD LEAD TO INCREASED LOGIC DESIGN DIFFICULTY (NON-FUNCTIONAL)

85451-28

Figure A-33.

HUGHES

MANUFACTURERS PROJECTIONS FOR GATES/CHIP ARE BEING VERIFIED



85451-45

Figure A-34.

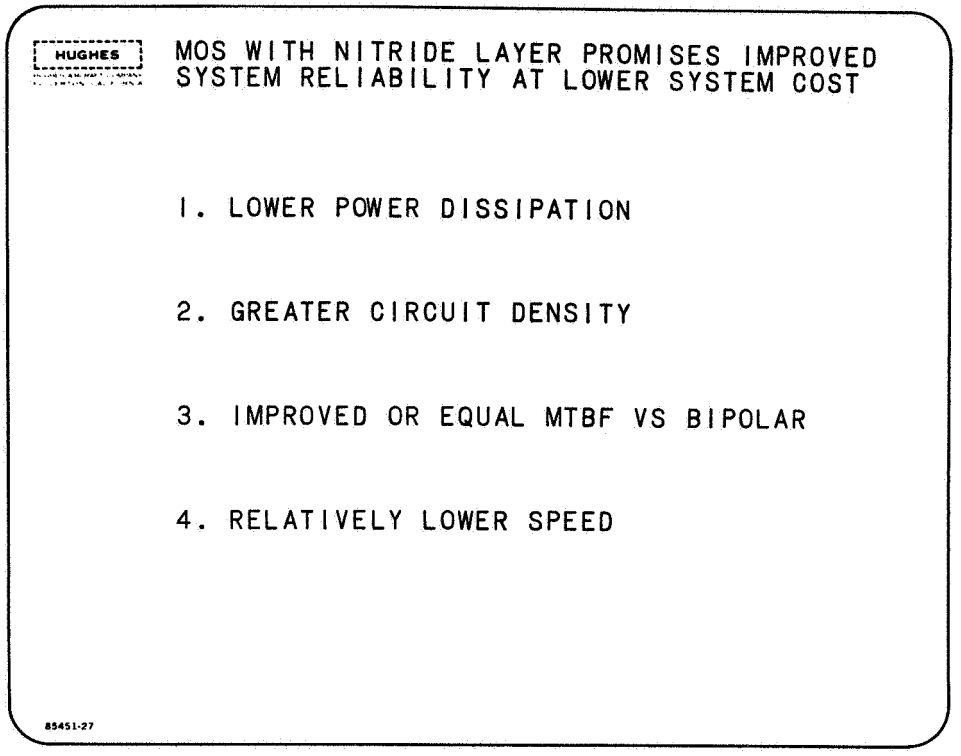


Figure A-35.

HUGHES BASED ON FAN-IN OF 2.8 AND A FAN-OUT OF 4

MOS GATE DELAY
IN NANOSECONDS

TYPE	CONVENTIONAL MOS	ION IMPLANTED
P - CHANNEL	87	16
N - CHANNEL	43	8
COMPLEMENTARY	13	3

85451-48

Figure A-36.

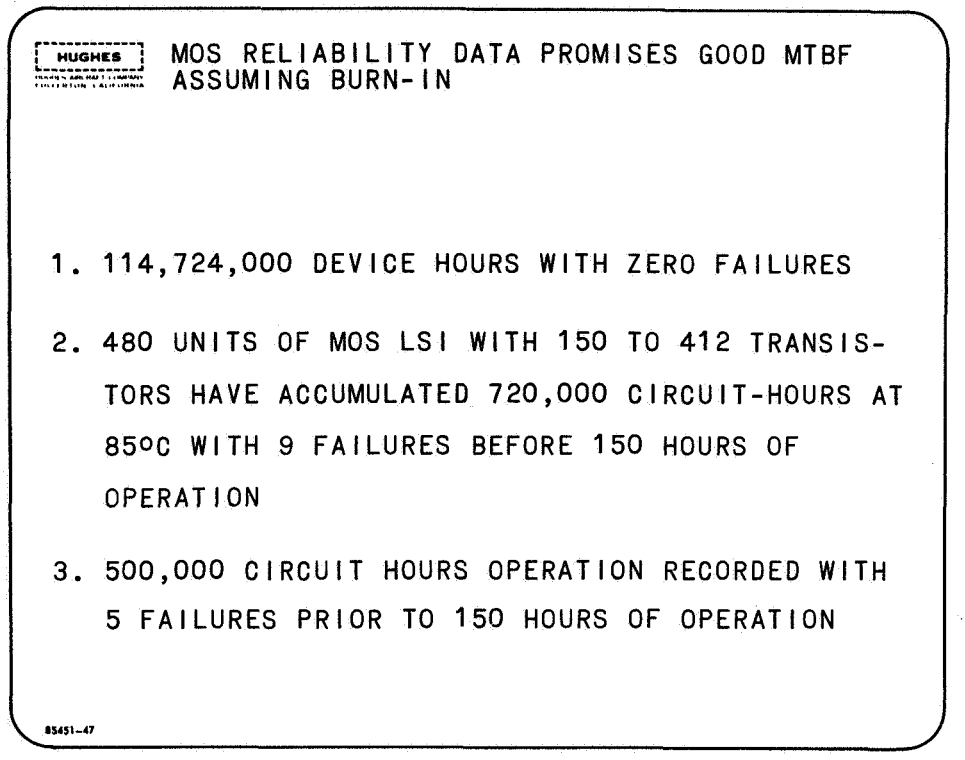


Figure A-37

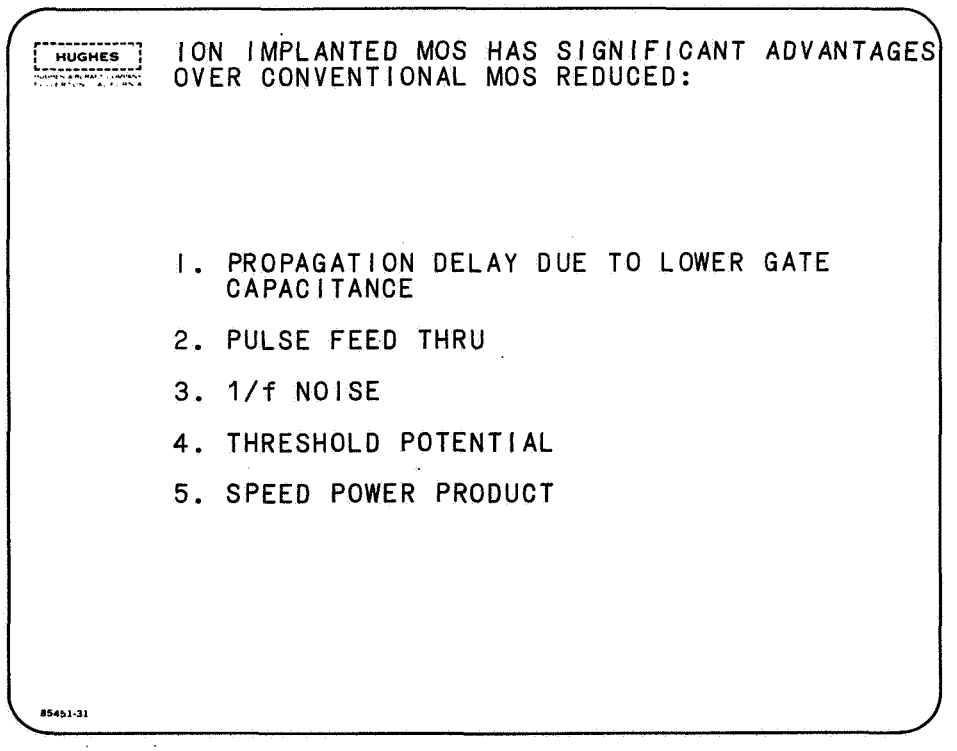
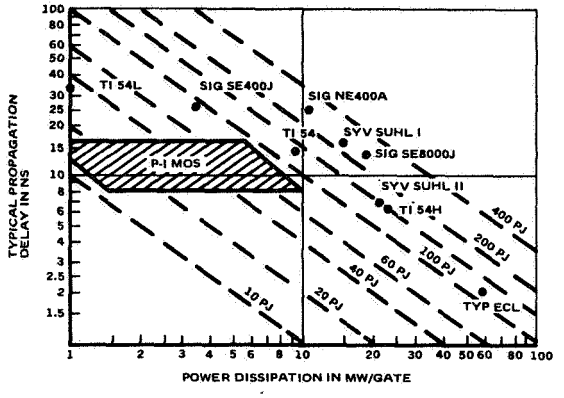


Figure A-38.

HUGHES
AIRCRAFT COMPANY
GROUND SYSTEMS GROUP

SPEED – POWER PRODUCT GRAPH ILLUSTRATES ADVANTAGE OF CUSTOM DESIGNED IMOS (P-CHANNEL) OVER LEADING TTL CIRCUITS



92602-49

Figure A-39

HUGHES
HUGHES AIRCRAFT COMPANY
FULLERTON, CALIFORNIA

IMPLEMENTATION COMPARISON SHOWS FUNCTIONAL CHARACTERS EFFECTIVE

COMPARISON OF LOGIC IMPLEMENTATION

IMPLEM / ITEM	FUNCTIONAL CHAR. FIRST IMPLEMENT.	FUNCTIONAL CHAR. IMPROVED IMPLEMENT.	PERCENT - MCB REF FIRST IMPLEMENT.	PERCENT - MCB REF IMPROVED IMPLEMENT.	MCB
TYPES	10	11	44	48	23
CARDS	233	172	40	31	554
PINS	23,600	19,700	50	41	47,600
GATES	47,200	47,200	135	135	35,000
G/P	2.0	2.4	267	320	0.75
COMPARISON OF PERFORMANCE					
GIBSON MIX INSTRUCTION	6.33/5.44 µs 10.30* µs		68/58 111*		9.30 µs

*FIRST PASS, 20 MHz

85452-30

Figure A-40.

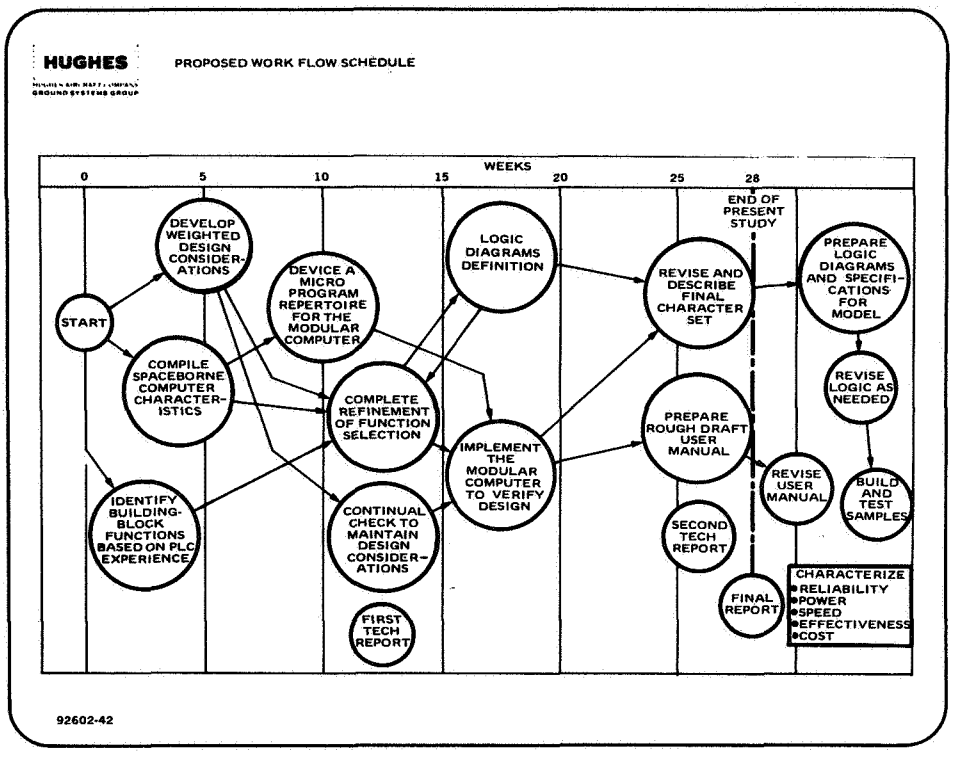


Figure A-41

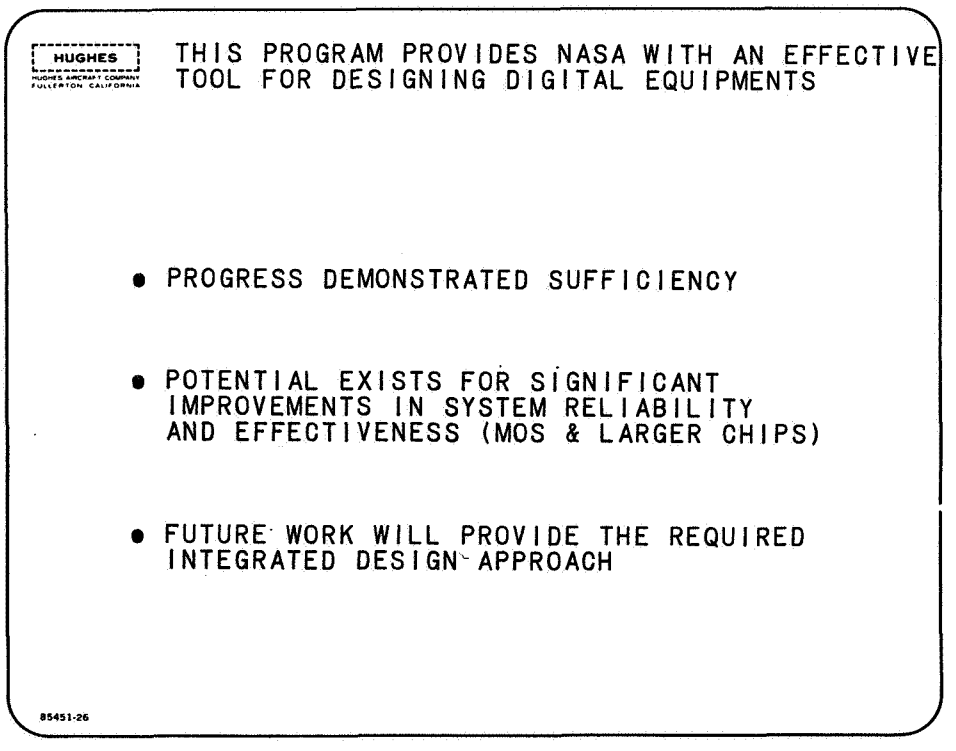


Figure A-42

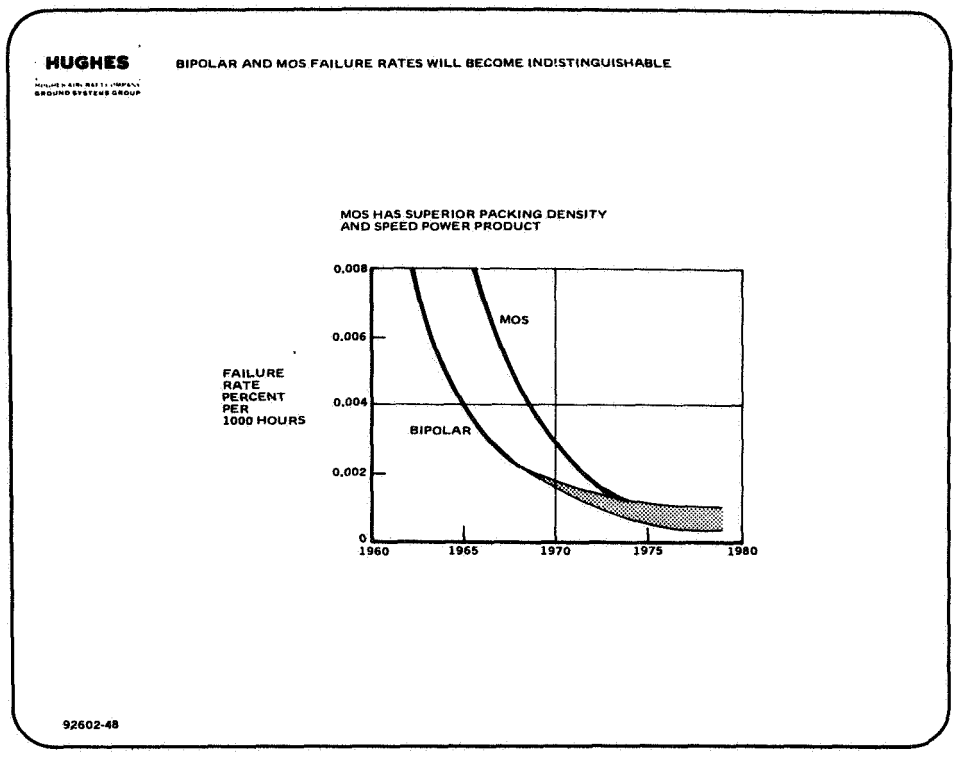

HUGHES
GROUND SYSTEMS GROUP
BIPOLAR AND MOS FAILURE RATES WILL BECOME INDISTINGUISHABLE
MOS HAS SUPERIOR PACKING DENSITY
AND SPEED POWER PRODUCT
FAILURE
RATE
PERCENT
PER
1000 HOURS
0.008
0.006
0.004
0.002
0
MOS
BIPOLAR
1960
1965
1970
1975
1980
92602-48

Figure A-43